COMP311: *COMPUTER ORGANIZATION!*

Lecture 5: Logic Gates

tinyurl.com/comp311-sp26

Logistics Updates

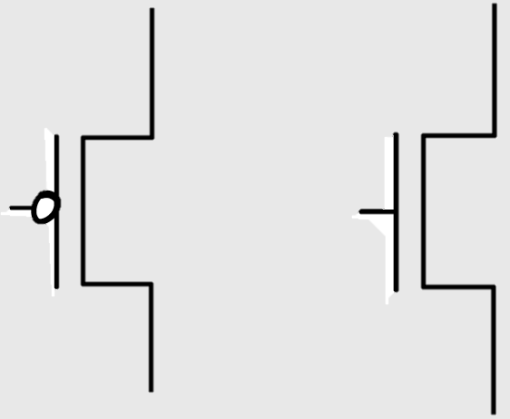
- WA1 due tonight!
  - *WA2 to follow shortly*
- Quiz schedule will be updated. I will announce via Canvas

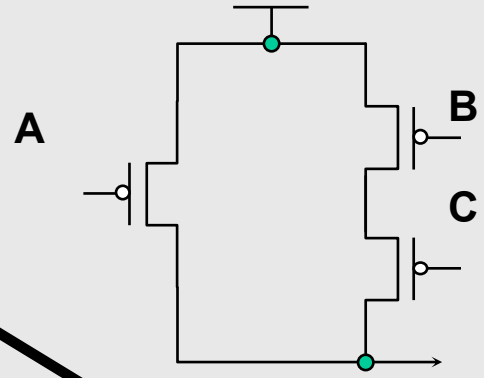
BUILDING LOGIC GATES WITH TRANSISTORS

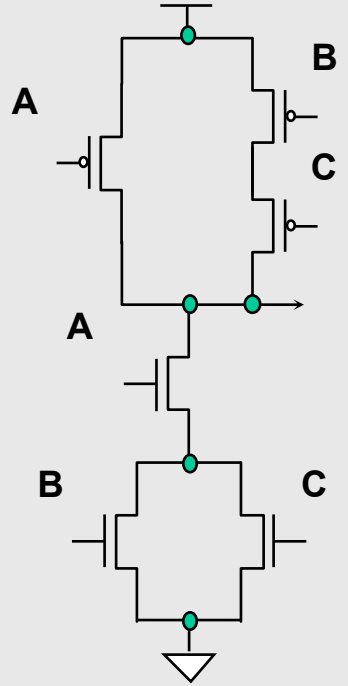
Abstraction ☺

CMOS Gates

Physical Transistors







CMOS gates abstract away individual transistors, letting us design reliable logic at a higher level.

CMOS gates are the abstraction that lets many transistors behave like a single logical unit.

CMOS Gates

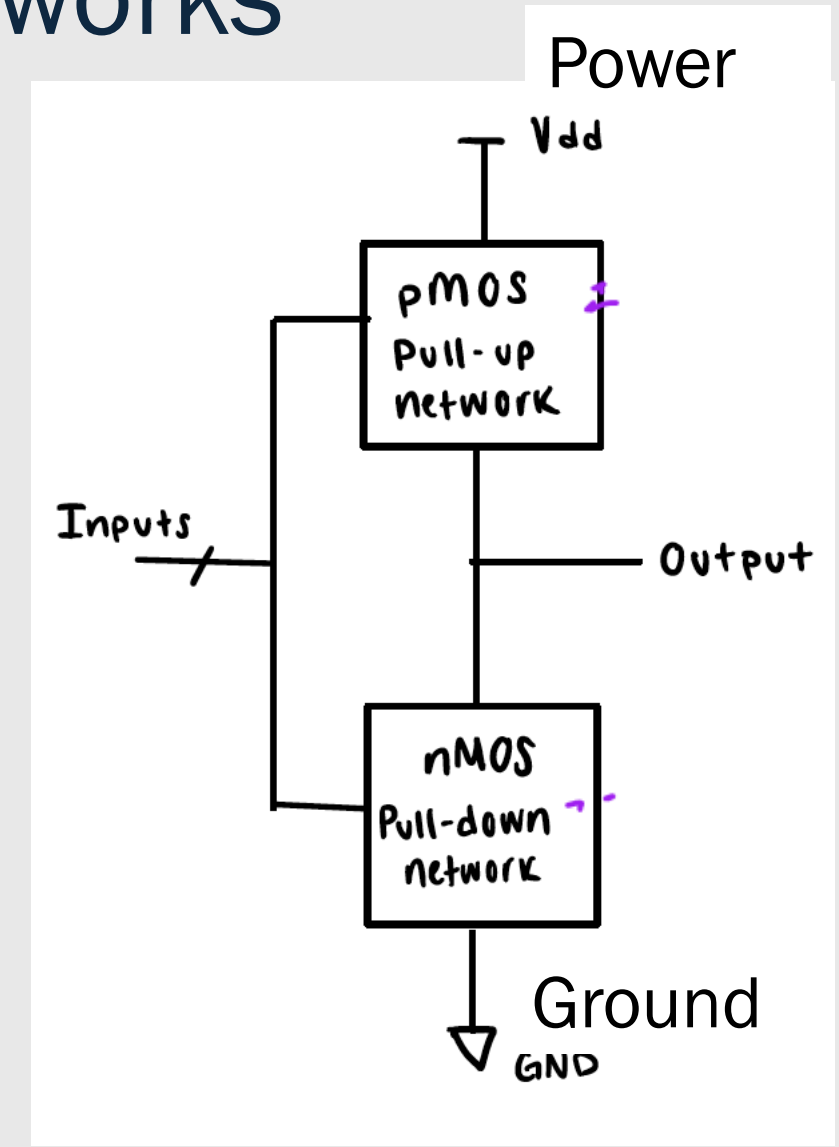
- CMOS = Complementary MOS
  - *The pMOS and nMOS transistors complement each other to form the logic gate*
- Common confusion
  - *CMOS is not a type of transistor!*
  - *It is a technology that combines both nMOS and pMOS transistors*

***MOS*: Metal Oxide Semiconductor ☺**

You might also see ***MOSFET:*** Metal Oxide Semiconductor Field-Effect Transistor

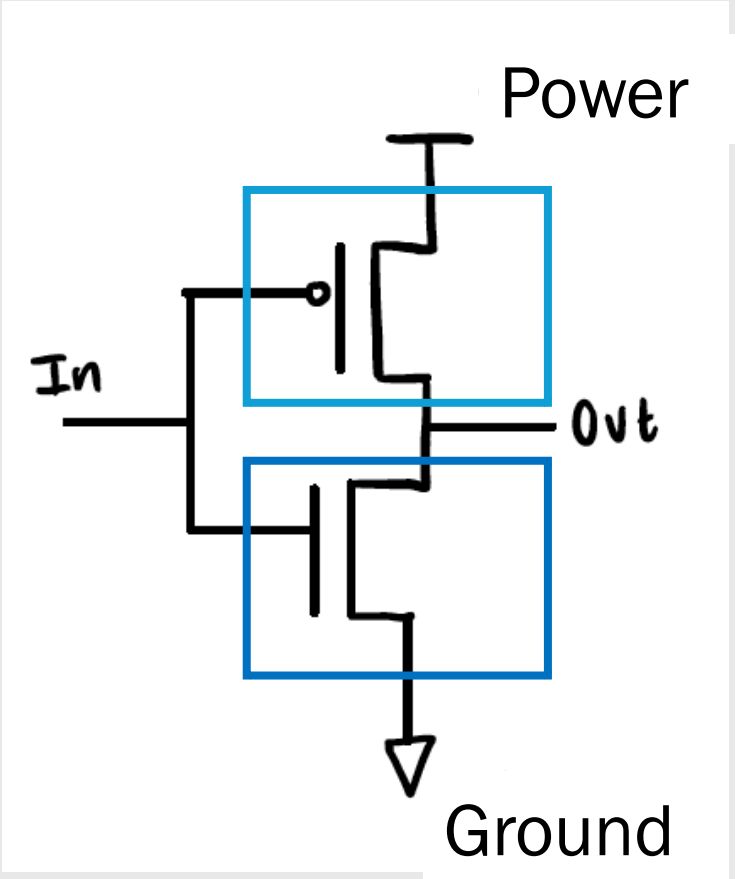
Pull-up and Pull-down Networks

- Only one network will be on at a time
- The pull-up network drives the output when the output is 1
- The pull-down network drives the output when the output is 0



CMOS Inverter

In	Out
0	1
1	0



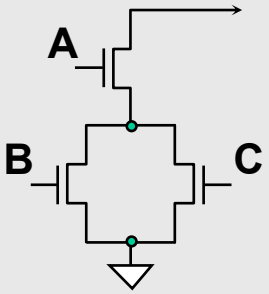
pull-up network

pull-down network

General CMOS Gate Recipe

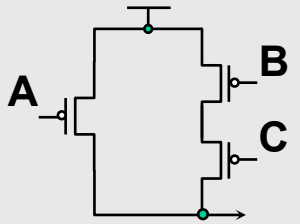
Step 1. Figure out pulldown network that does what you want (i.e the set of conditions where the output is '0')

e.g., F = A*(B+C)



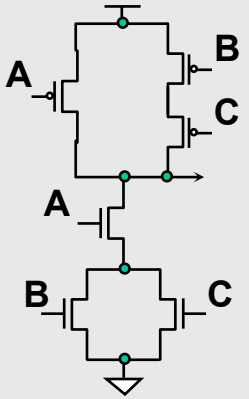
Recall:
Series = *AND*
Parallel = *OR*

Step 2. Build the "opposite" version of that network to figure out when the output should be 1.



This comes from DeMorgan's Laws! In practice, we swap series + parallel connections, use pMOS instead of nMOS

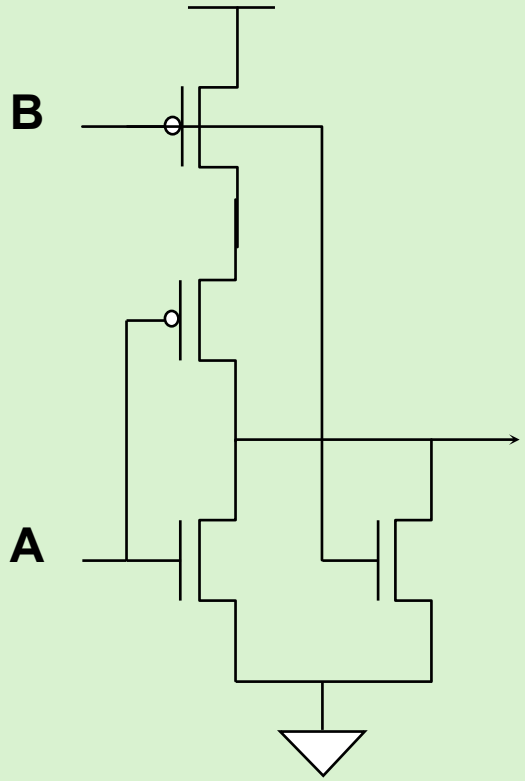
Step 3. Combine the two so you have a circuit that always pulls the output either up to 1 or down to 0.



But isn't it hard to wire it all up?

Yes it is! This does not scale...

What function does this gate compute?



NOR

A	B	C
0	0	
0	1	
1	0	
1	1	

Designing hardware at the transistor level…

- Does not scale!
- What should we do?
  - *Abstract again!*
    - Logic gates are next!
- But before we jump to the next level of abstraction, let's cover some ***standard***, ***easy-to-wire*** primitives
  - *NAND and NOR*
  - These functions are universal, meaning any Boolean function or digital logic can be built from them

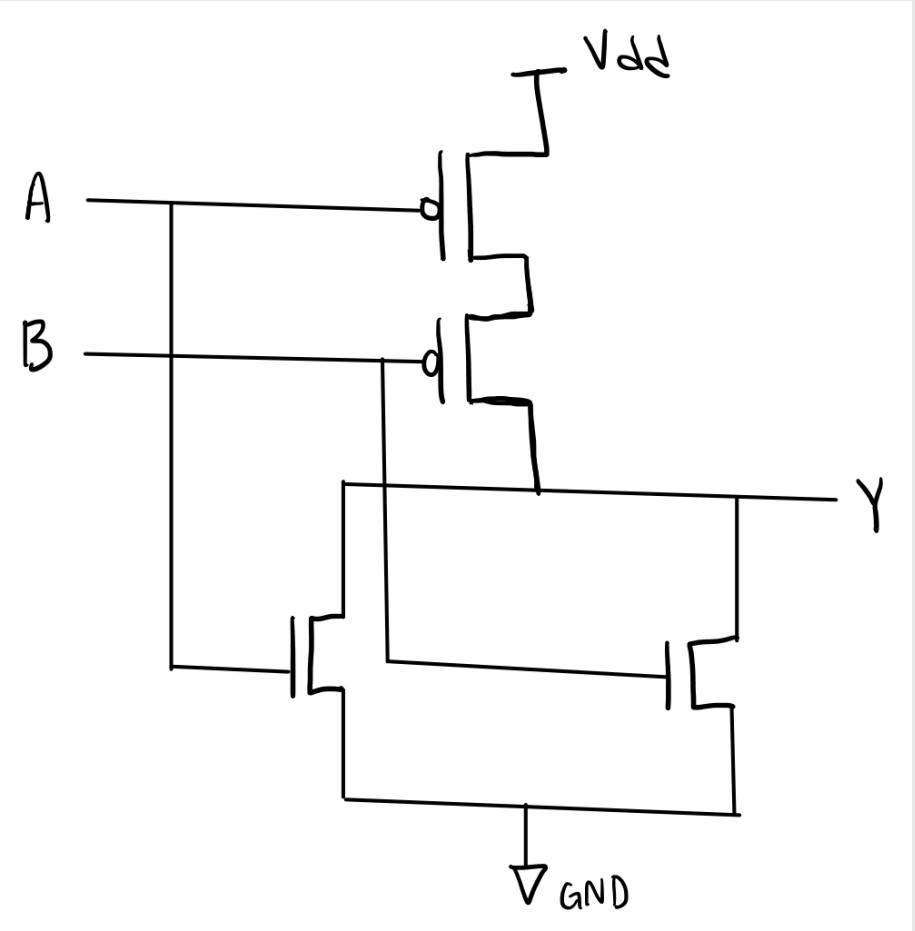
CMOS NOR

CMOS NOR

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

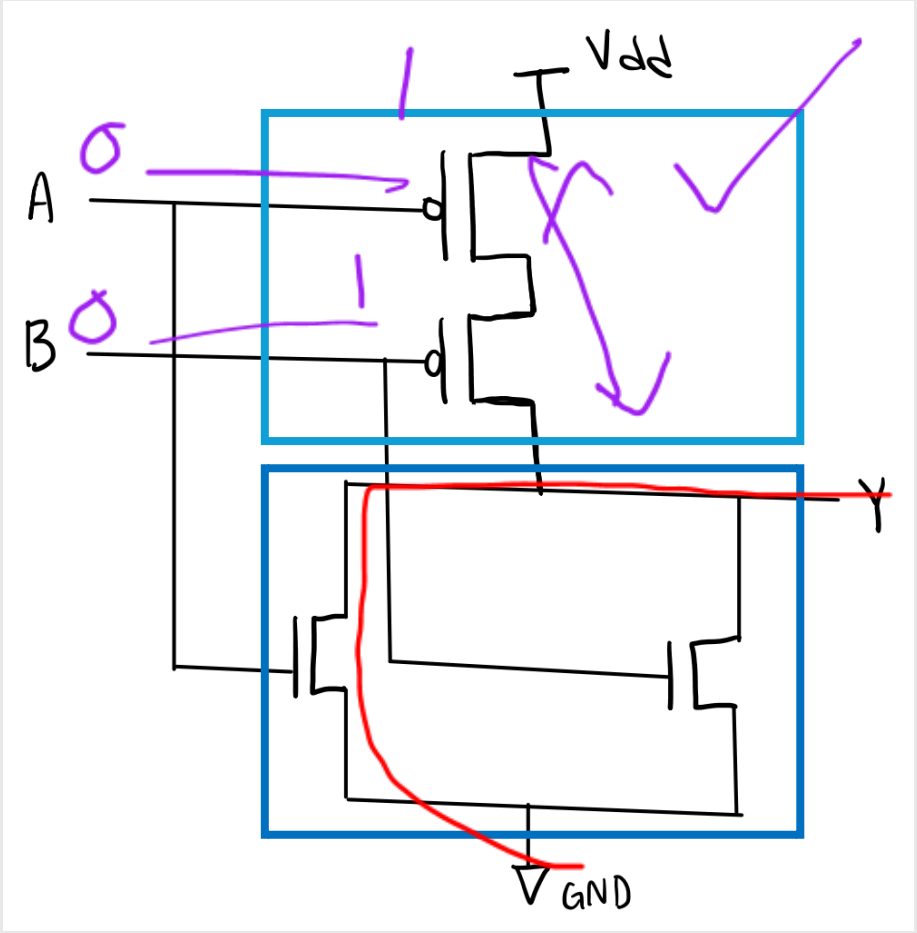
CMOS NOR

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0



CMOS NOR

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

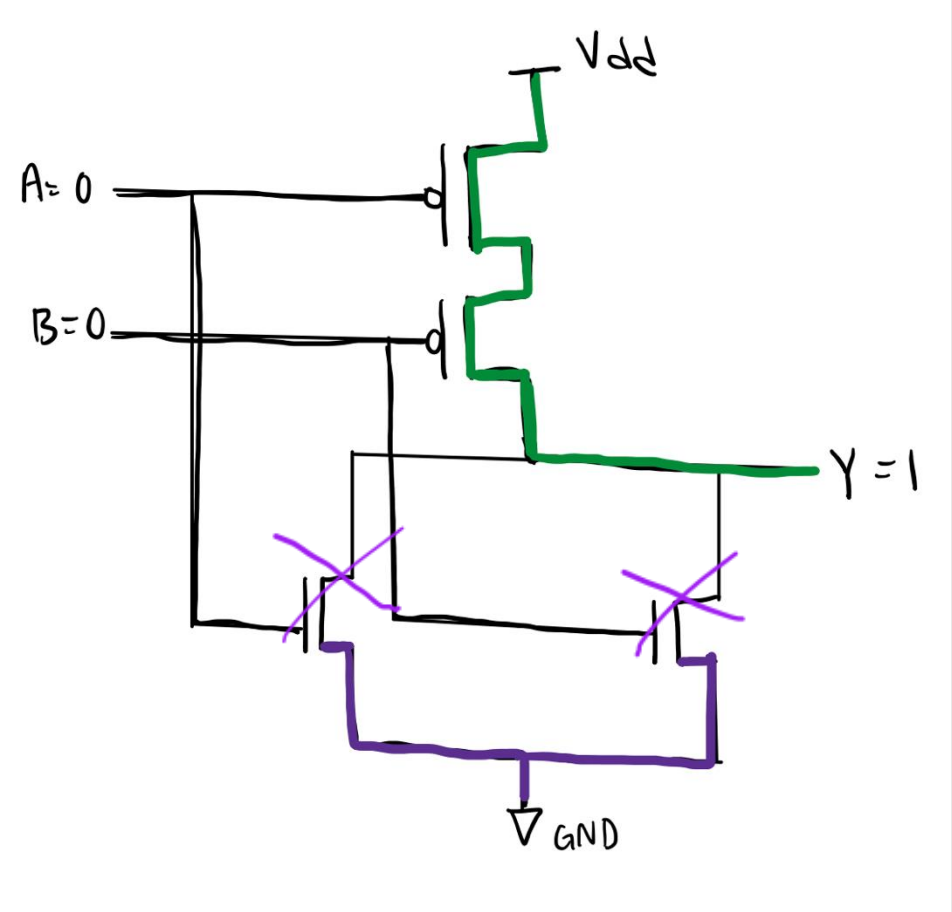


pull-up network

pull-down network

CMOS NOR

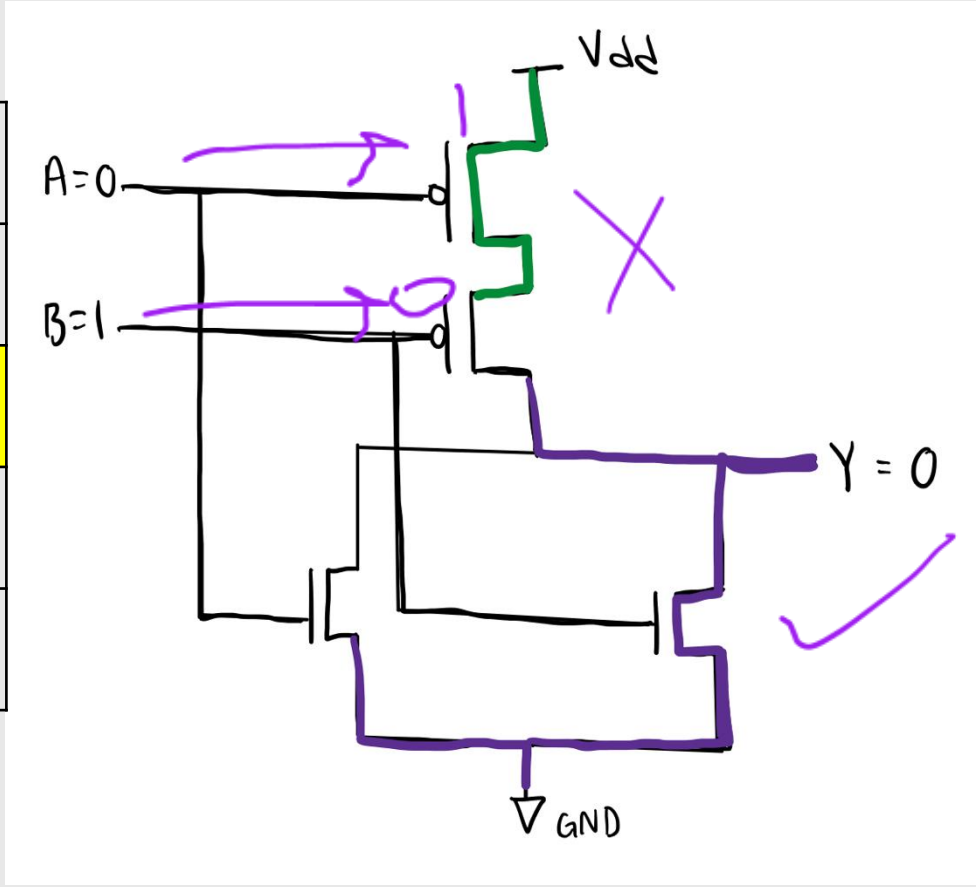
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0



both pMOS on, output pulled up

CMOS NOR

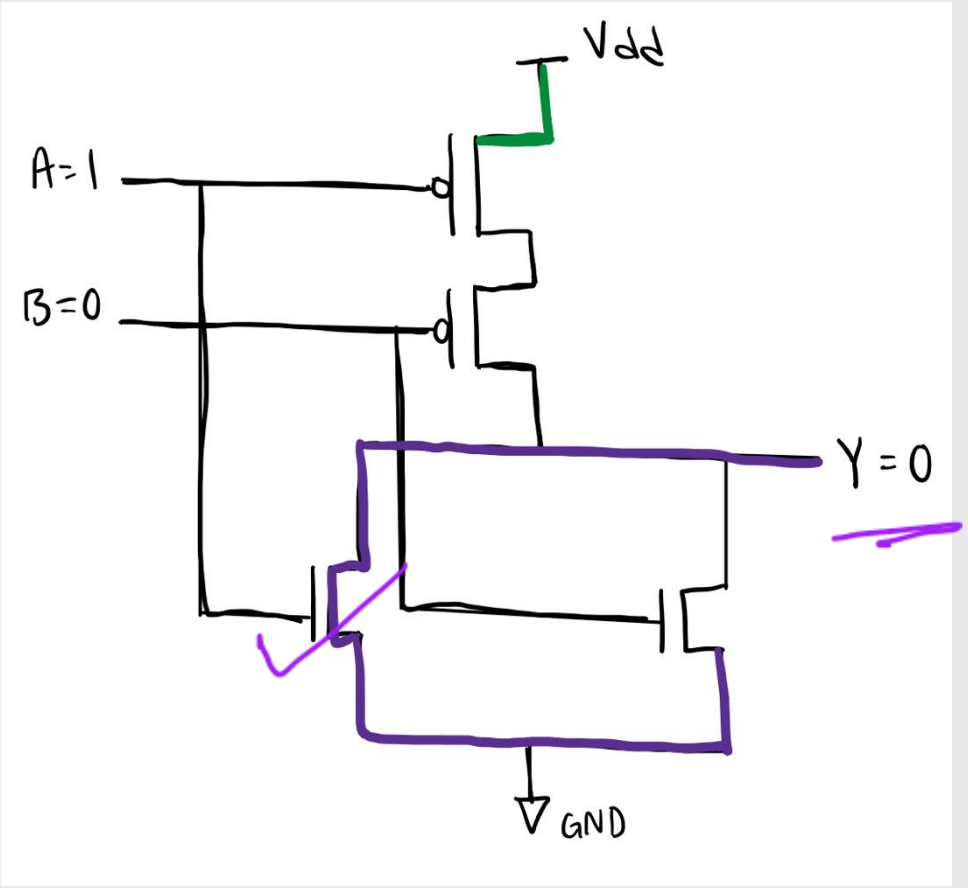
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0



B's nMOS on, pulls down

CMOS NOR

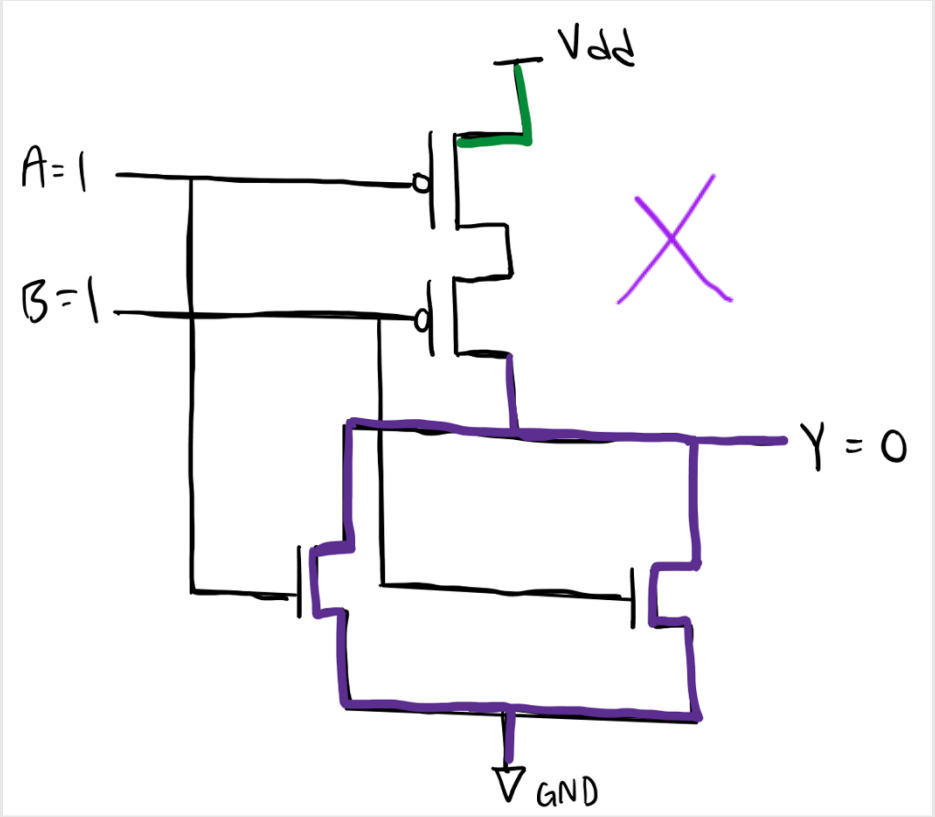
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0



A's nMOS on, pulls down

CMOS NOR

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0



both nMOS on, strong pull down

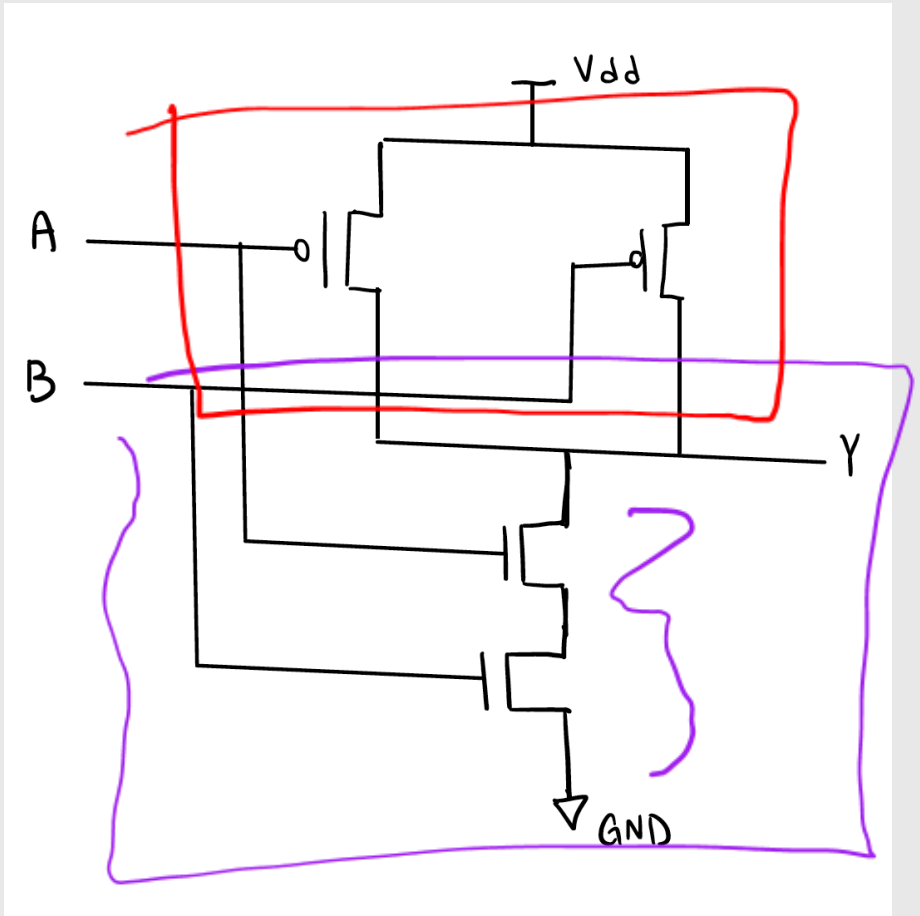
CMOS NAND

CMOS NAND

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

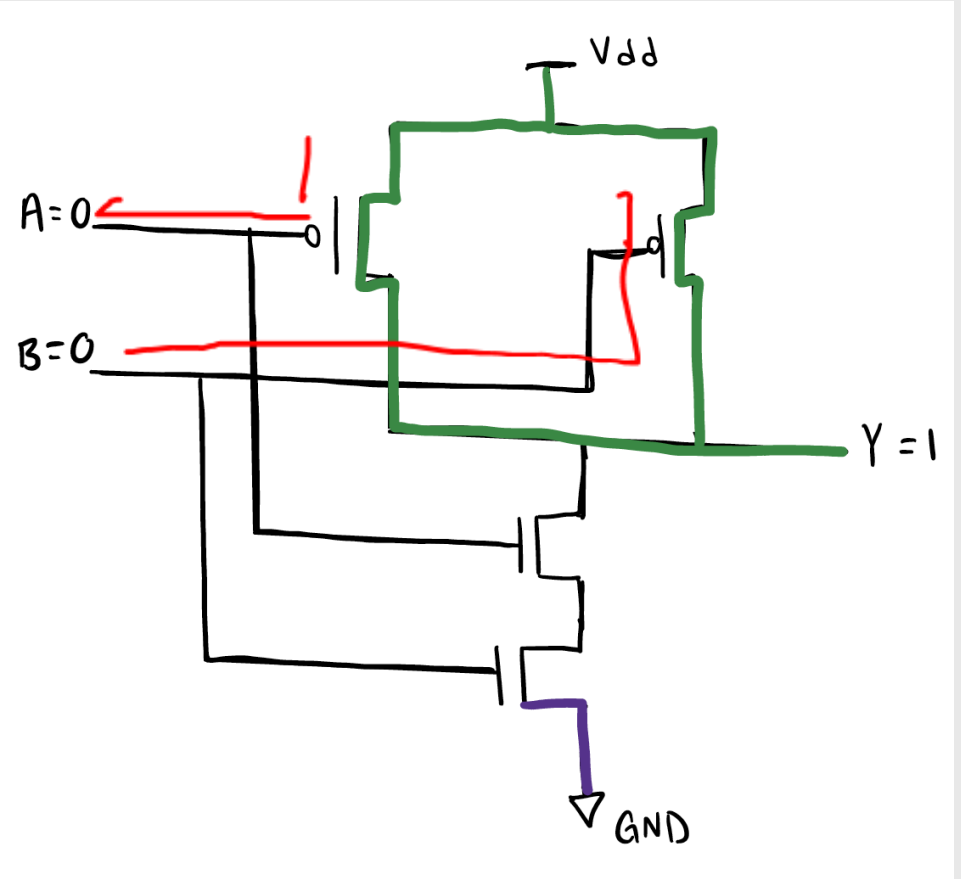
CMOS NAND

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



CMOS NAND

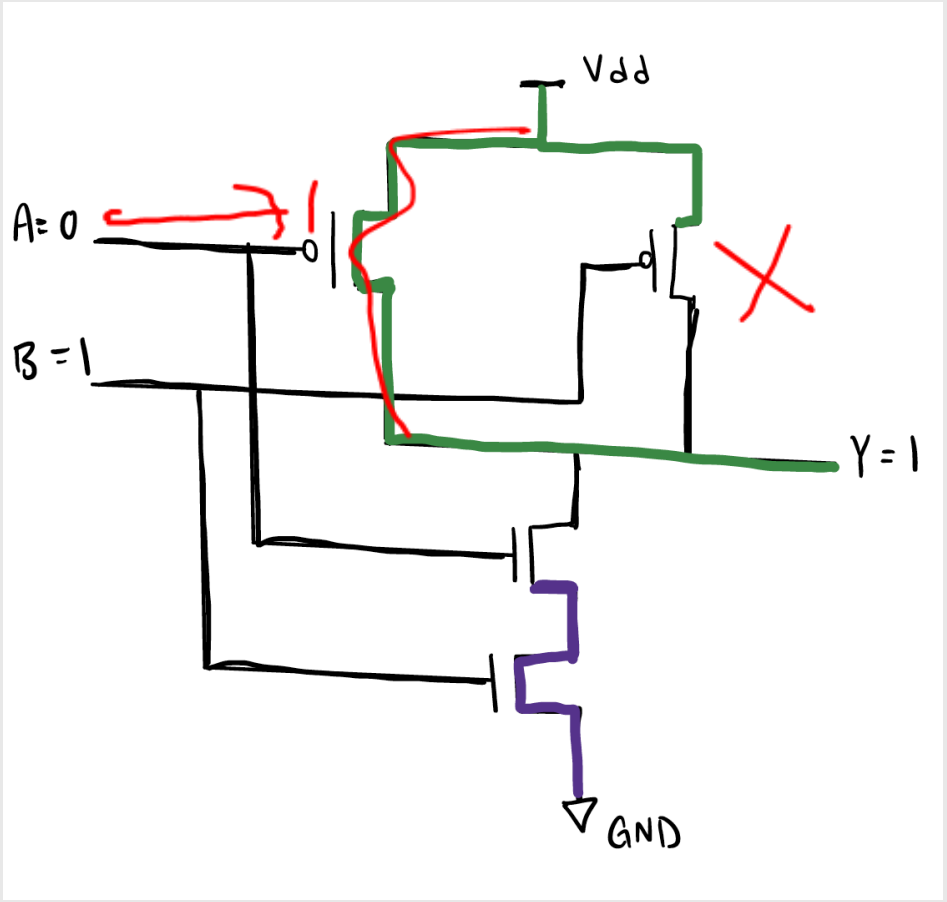
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



both pMOS on, pull-up works

CMOS NAND

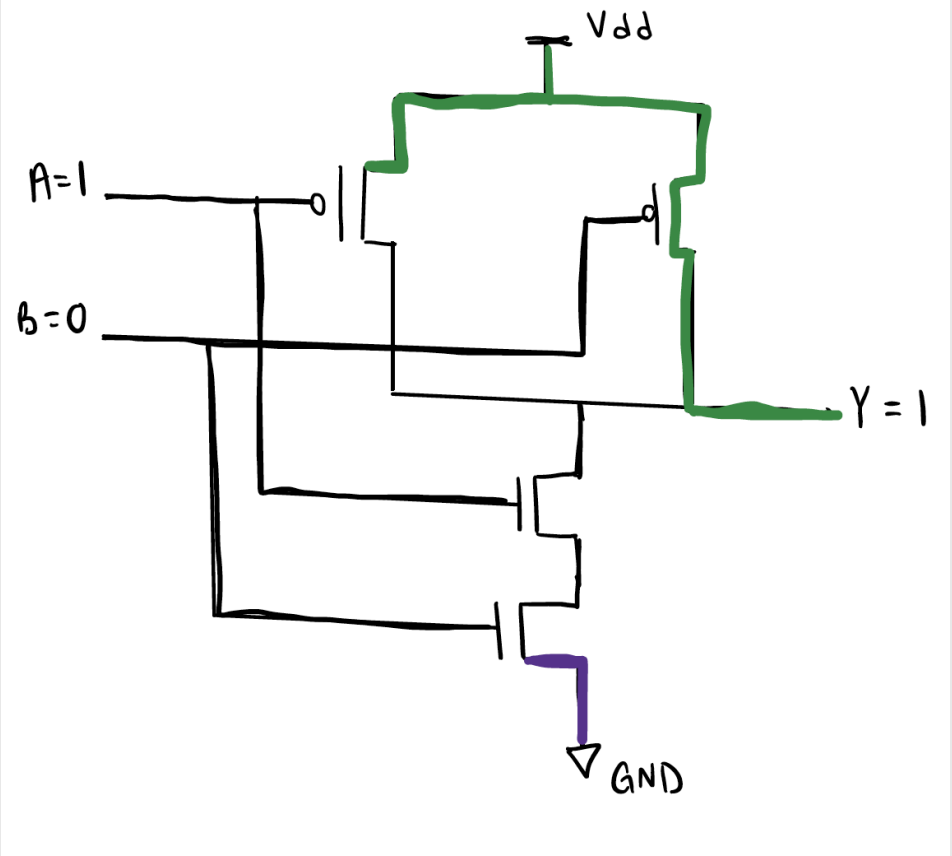
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



A's pMOS still on, output pulled high

CMOS NAND

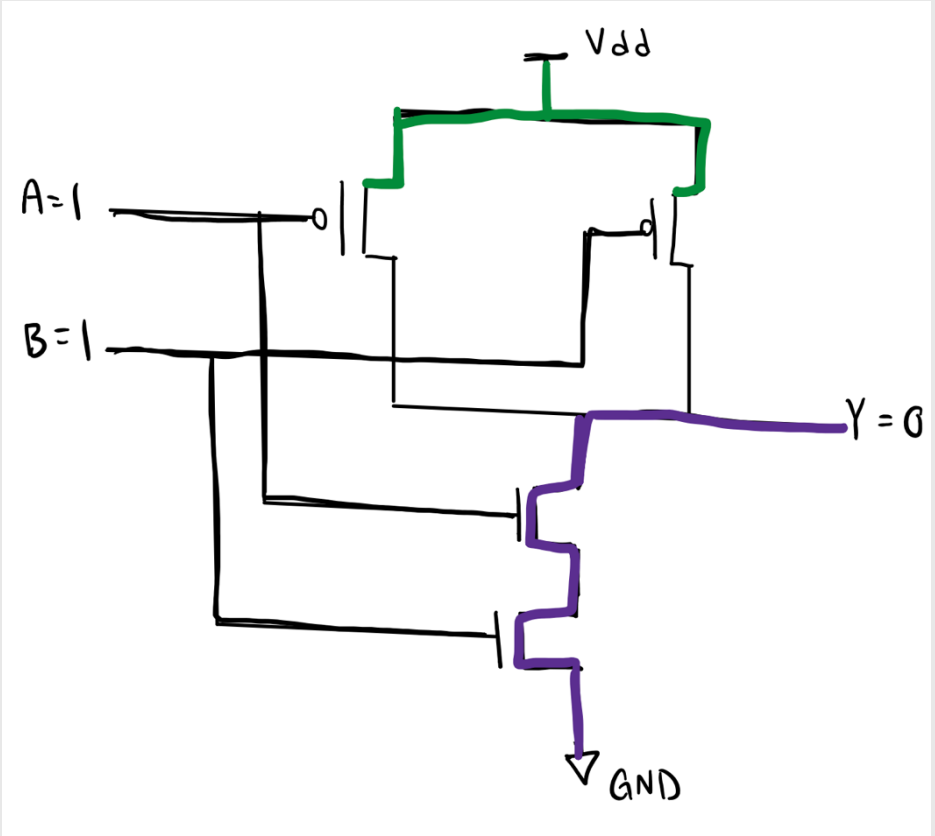
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



B's pMOS still on, output pulled high

CMOS NAND

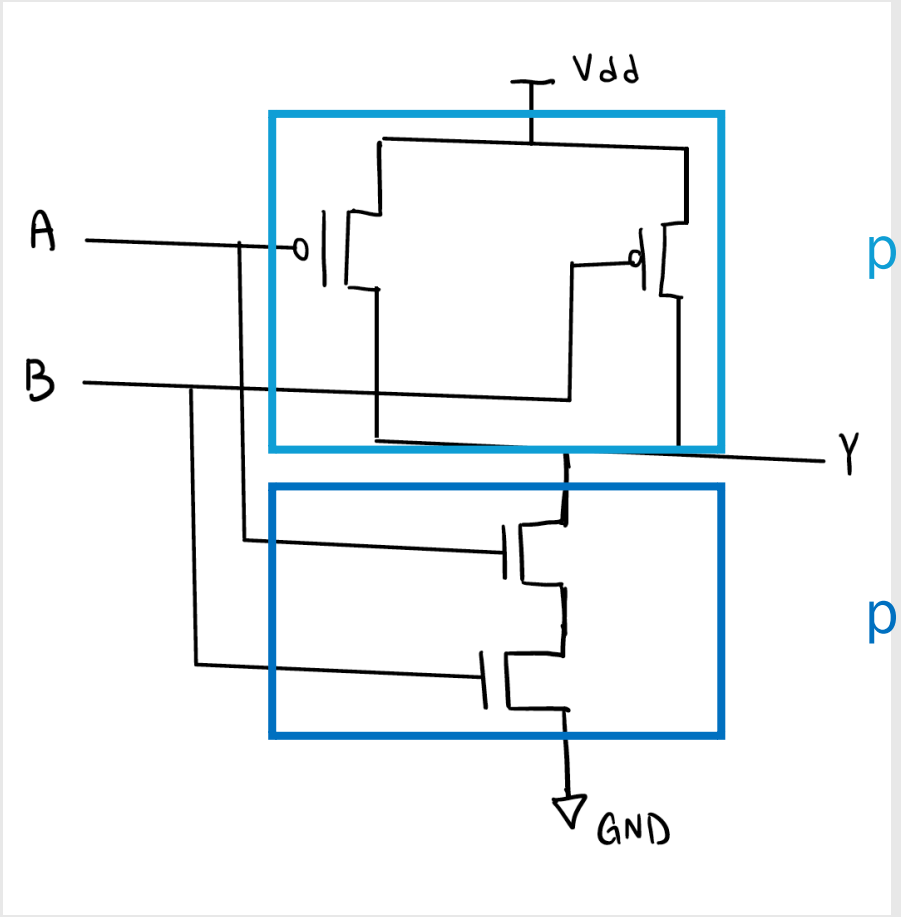
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



both pMOS off, both nMOS on → path to ground

CMOS NAND

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



pull-up network

pull-down network

CMOS AND

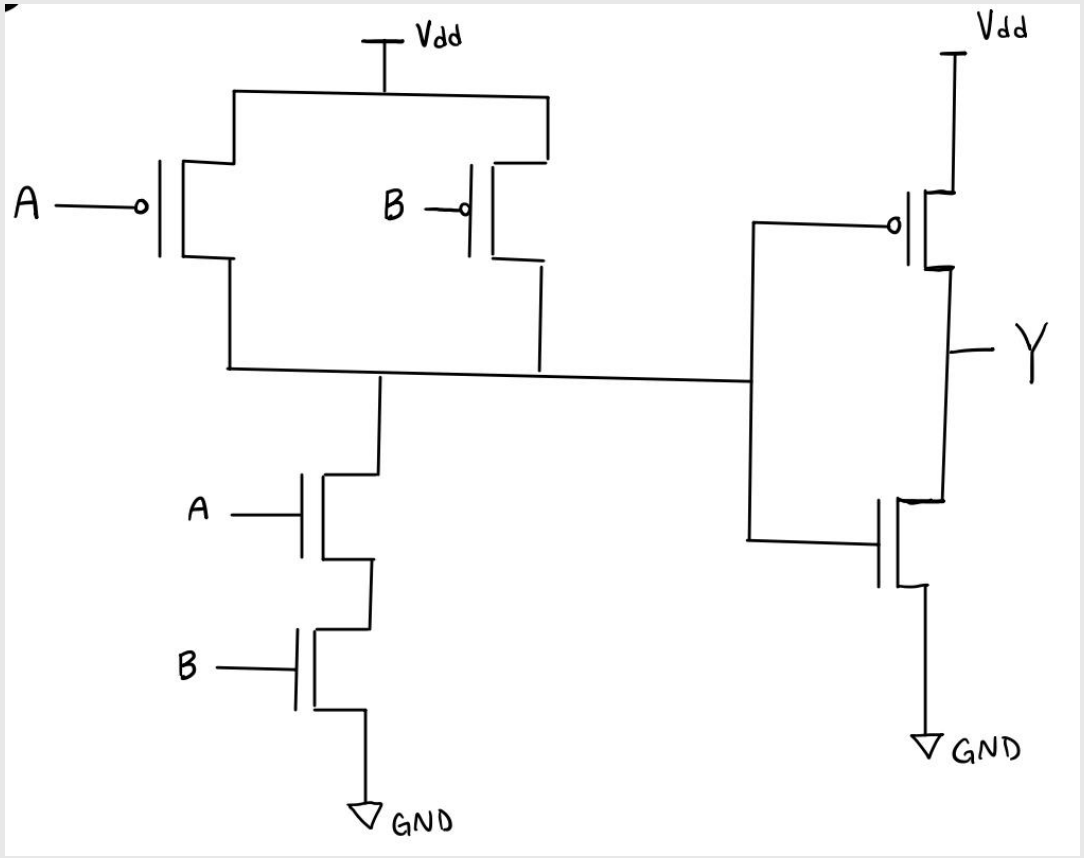
CMOS AND

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

CMOS AND

AND ≡ $\overline{\text{NAND}}$

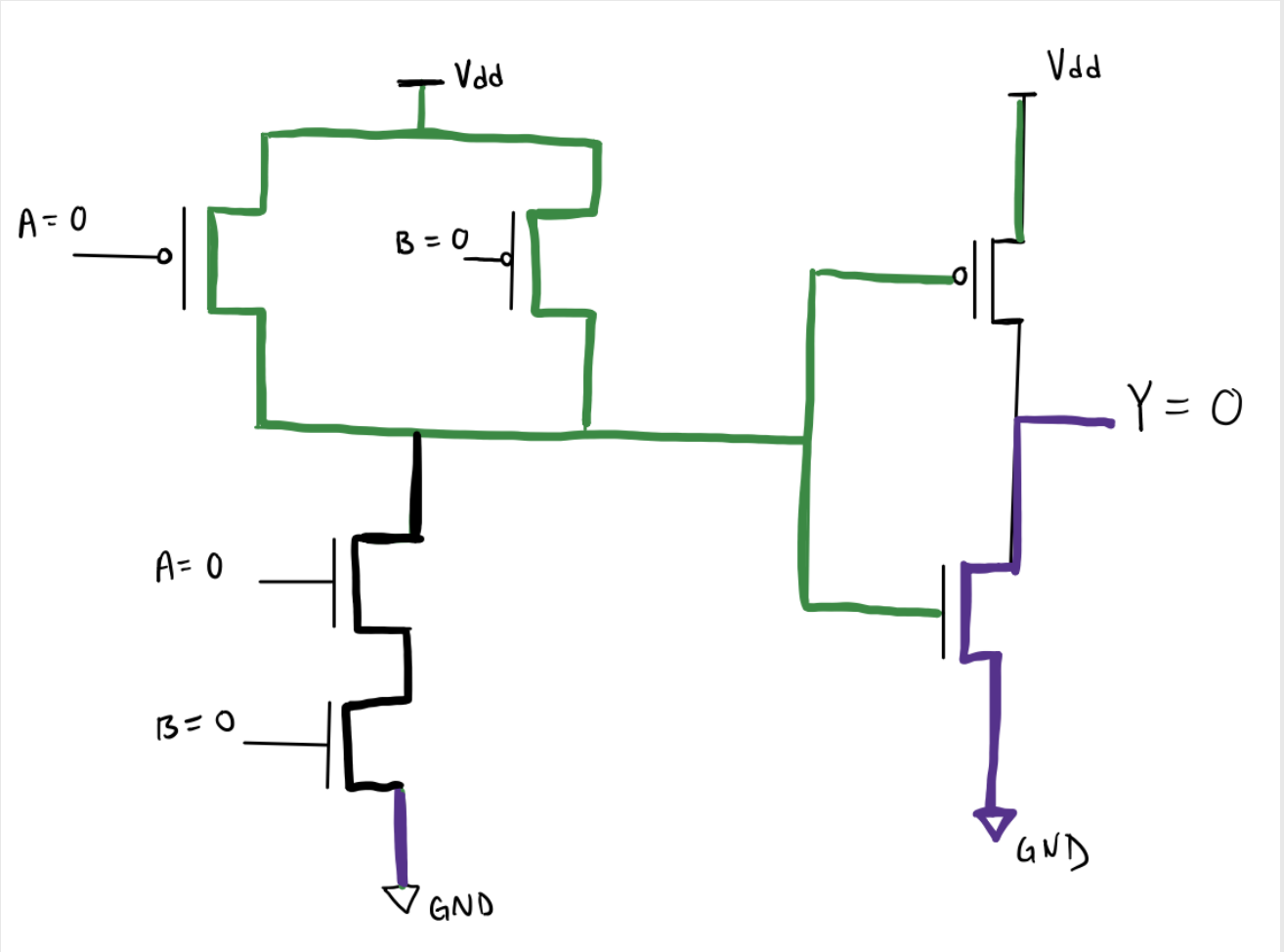
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



NAND INVERTER

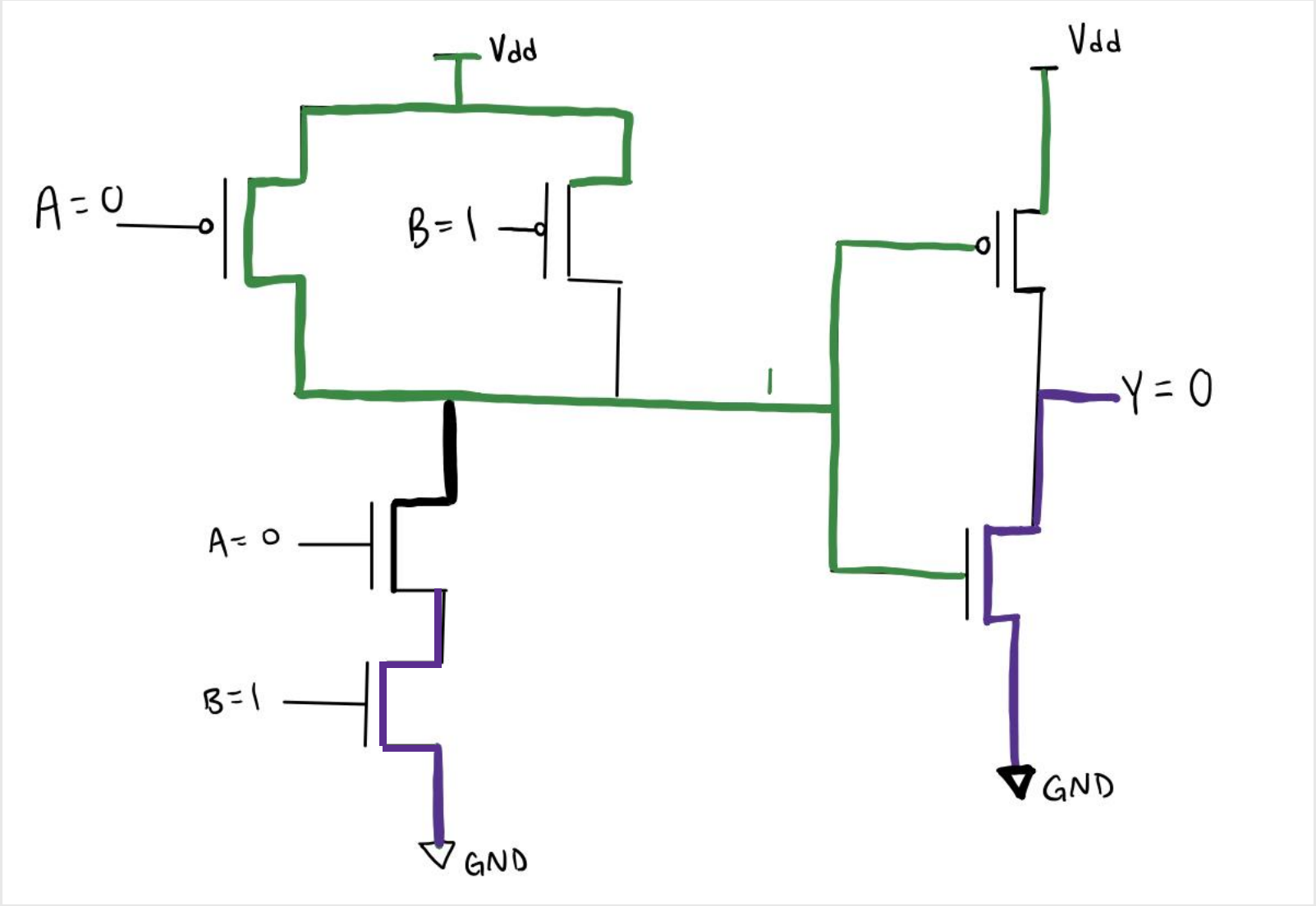
CMOS AND

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



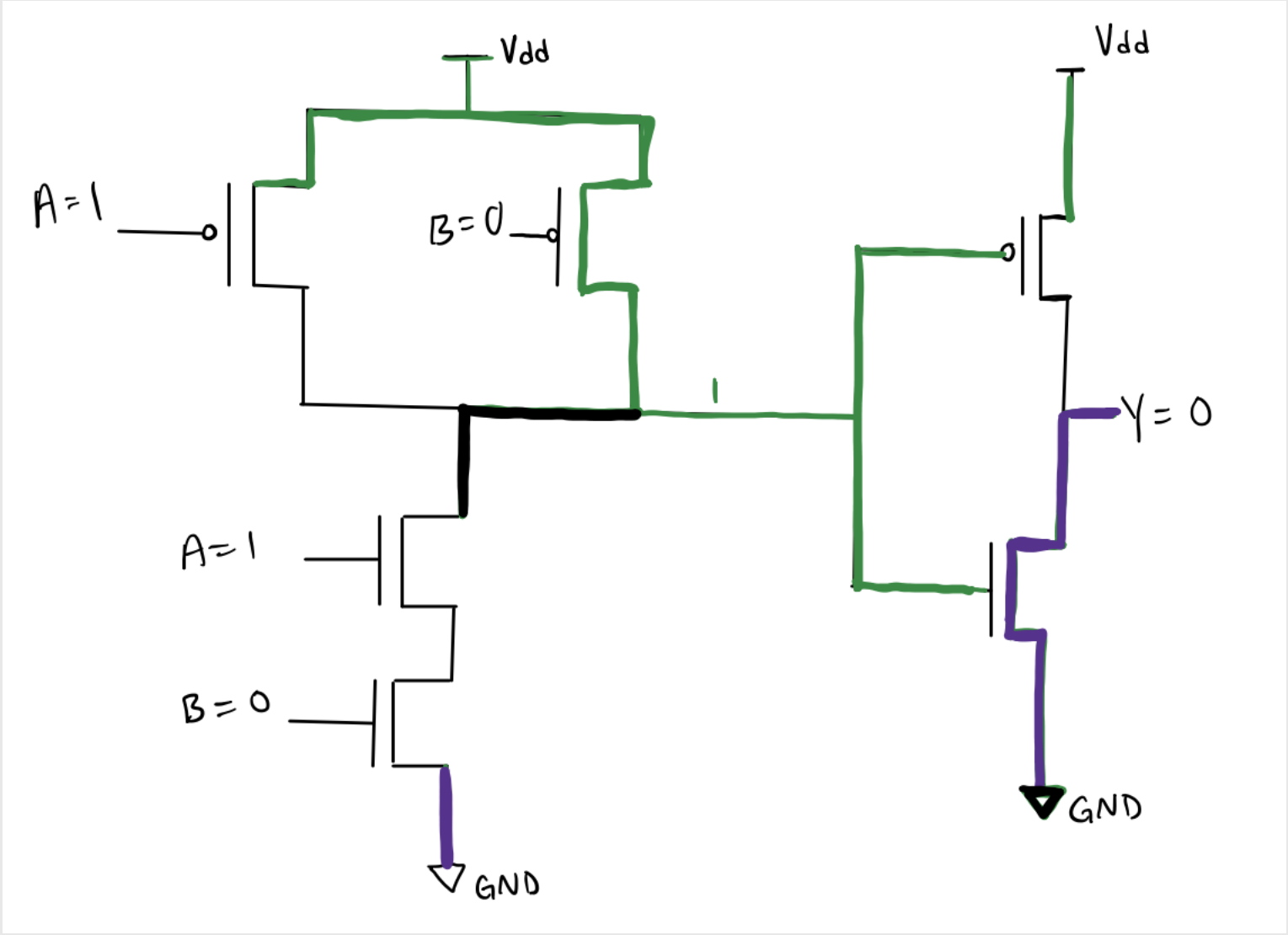
CMOS AND

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



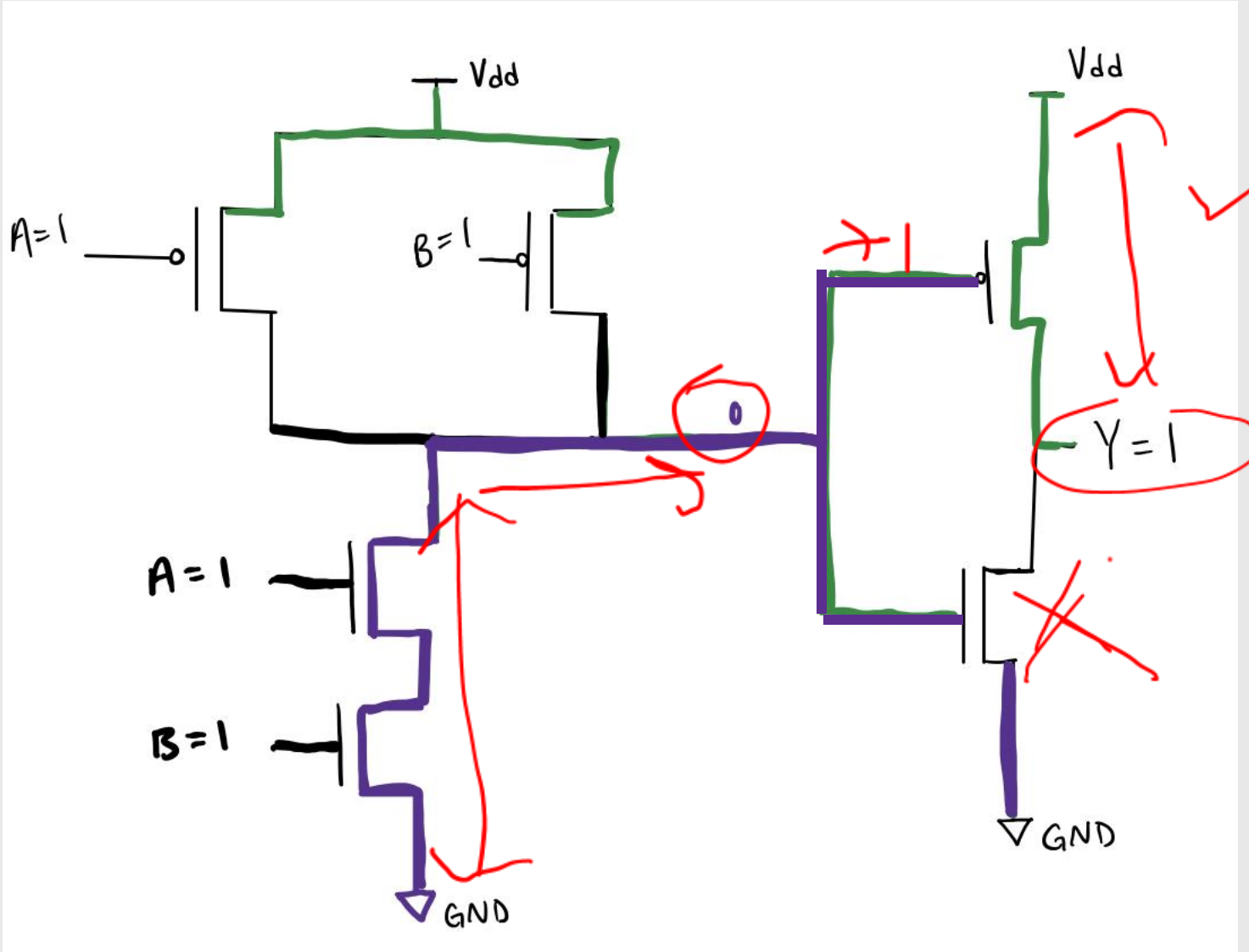
CMOS AND

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



CMOS AND

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



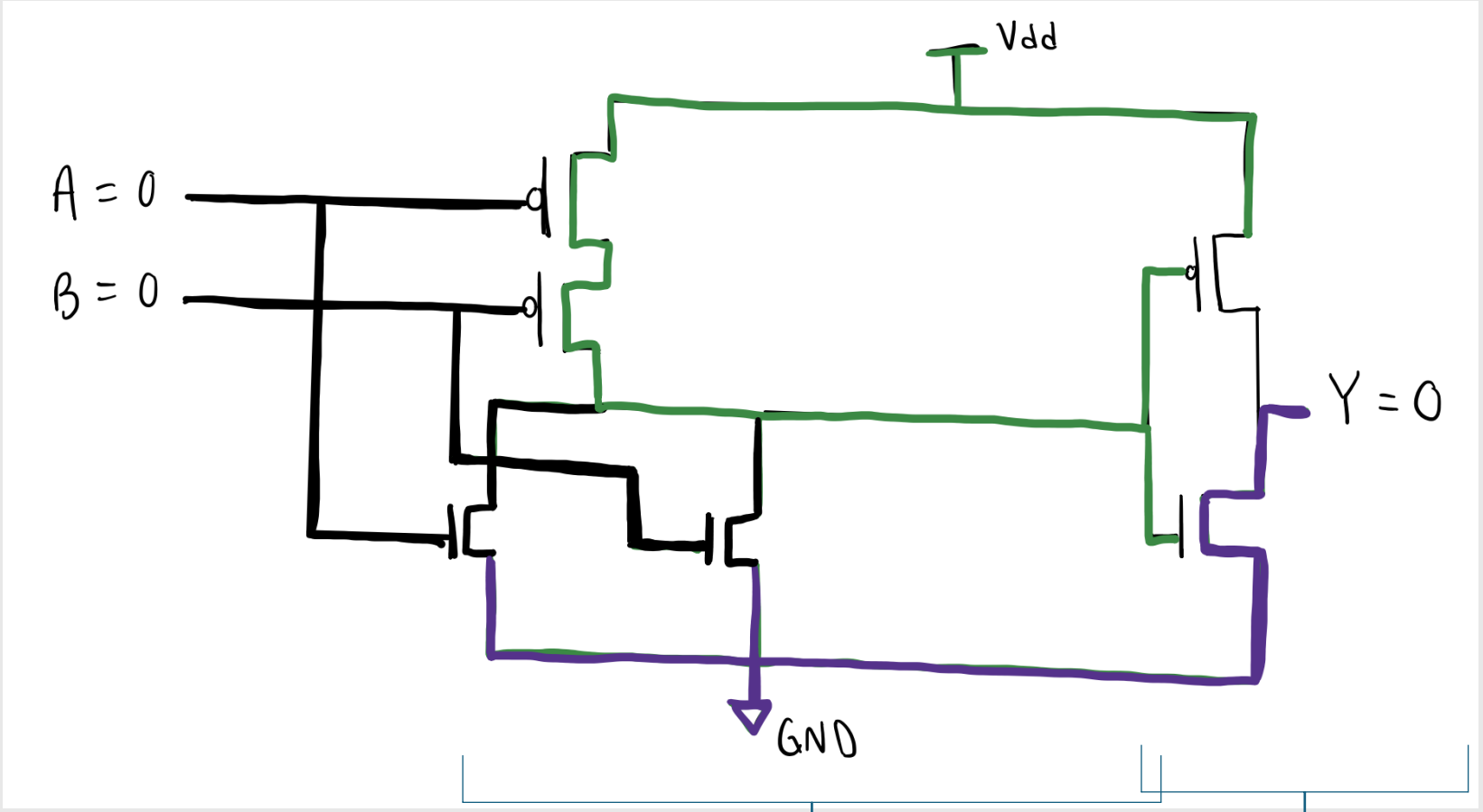
CMOS OR

CMOS OR

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

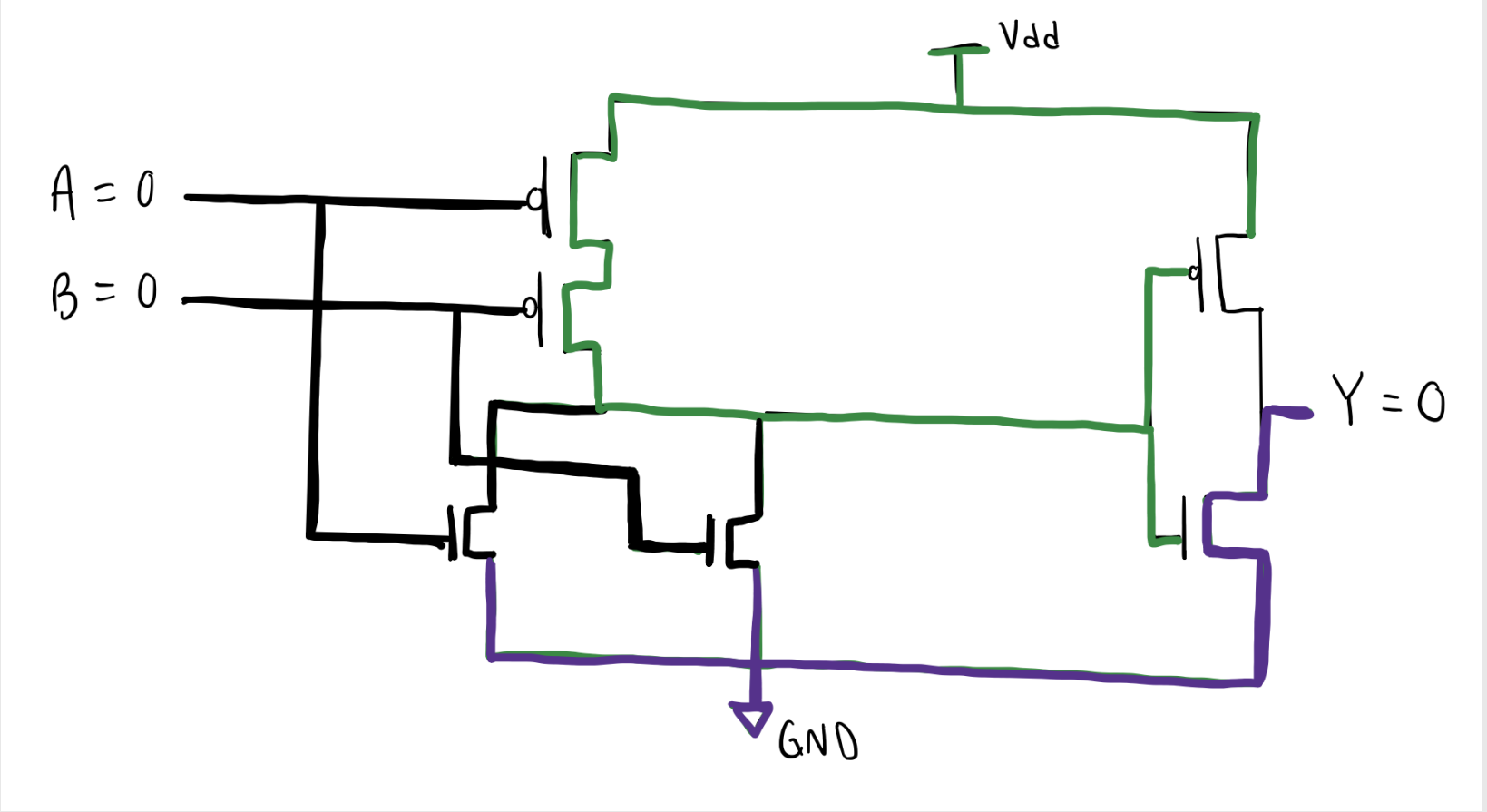
CMOS OR

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



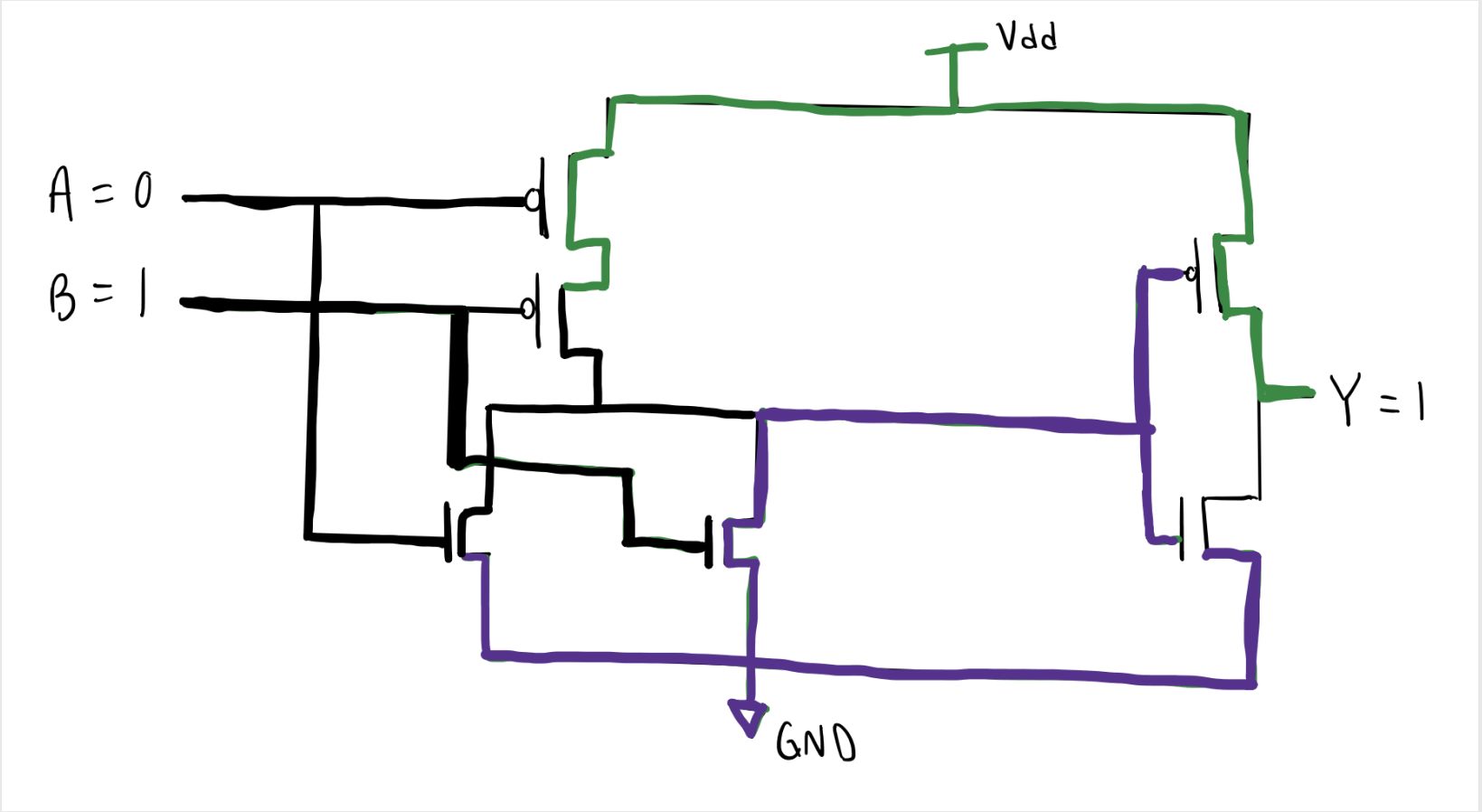
CMOS OR

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



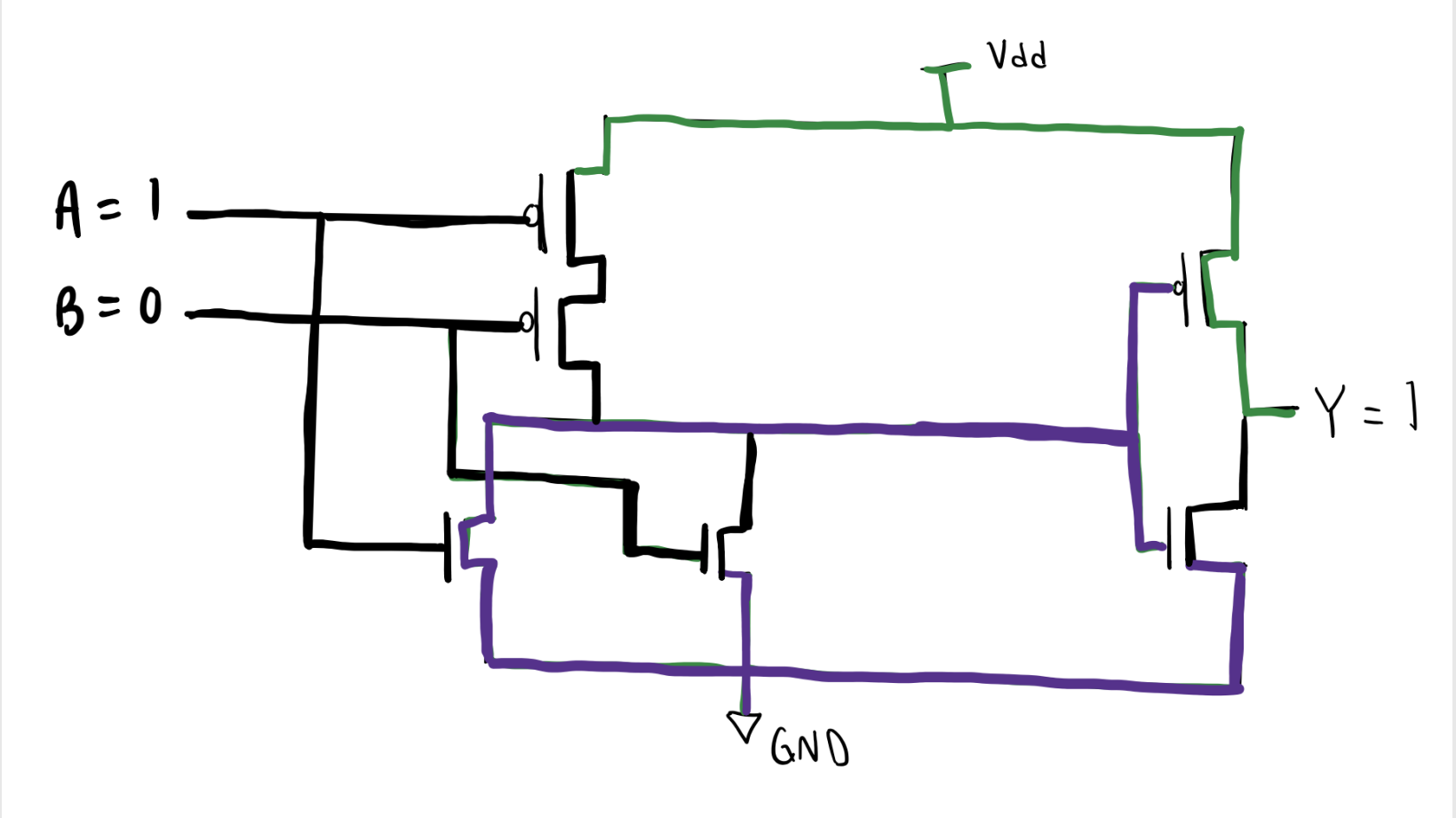
CMOS OR

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



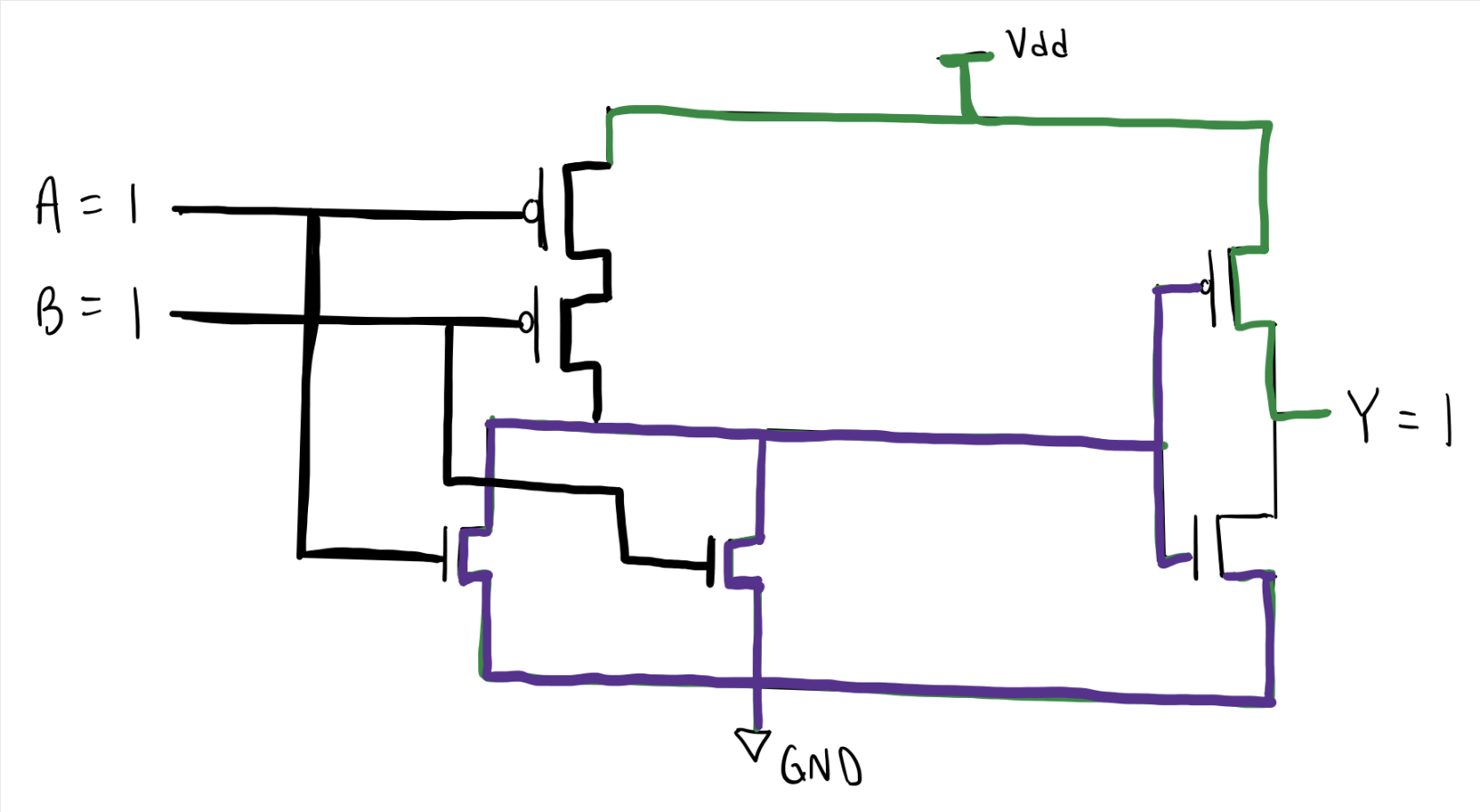
CMOS OR

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



CMOS OR

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



Transistor Count

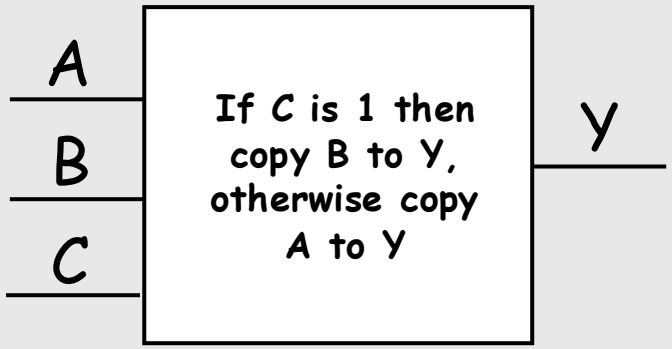
Gate	Number of Transistors
NOT	2
AND	6
OR	6
NAND	4
NOR	4
XOR	12
XNOR	12

LOGIC GATES

Popping up a level of abstraction! ☺

Now can we design larger systems!

We need to start somewhere – usually with a functional specification

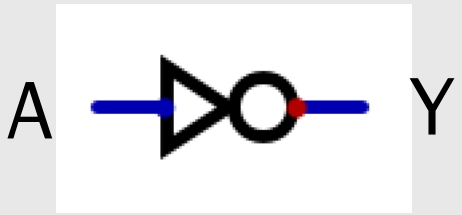


Every combinational function can be expressed as a table!

"**Truth tables**" are a concise description of the combinational system's function, where an output is specified for *every* input combination.

Inverter/Not Gate

Symbol



Circuit

Truth Table

A	Y
0	1
1	0

Equation

$$Y = \overline{A}$$

What logic gates can we build?

Recall, we need to design our gates using a pull-up network of pMOS transistors and a pull-down network of nMOS transistors

What gates can we
- build?
- define?

Let's start by considering only 2-input gates.

AND		OR		NAND		NOR	
AB	Y	AB	Y	AB	Y	AB	Y
00	0	00	0	00	1	00	1
01	0	01	1	01	1	01	0
10	0	10	1	10	1	10	0
11	1	11	1	11	0	11	0

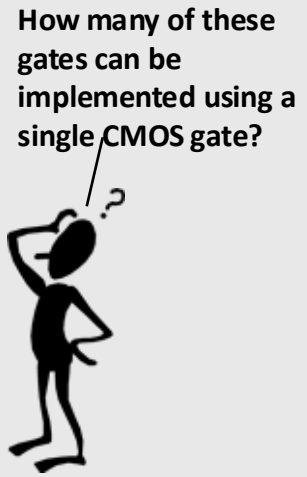
How many possible 2-input gates are there?

KEY IDEA: As many as there are 2-input truth tables.

All the gates!

There are only 16 possible 2-input gates… Let's examine all of them. Some we already know, others are just silly.

INPUT AB	ZERO	AND	A > B	A	B > A	B	XOR	OR	NOR	XNOR	NOT 'B'	A <= B	NOT 'A'	B <= A	NAND	ONE
00	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
01	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
10	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
11	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1



Do we really need all of these gates?

Nope! Once we realize that we can describe all of them using just AND, OR, and NOT

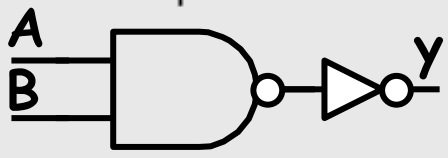
Composing gates: AND and OR

Each can be constructed using a pair of CMOS gates

AND is just NAND with an inverter, and OR is just NOR with an inverted output.

AND

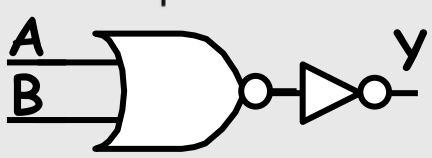
AB	Y
00	0
01	0
10	0
11	1



NAND

OR

AB	Y
00	0
01	1
10	1
11	1



NOR

These two gates are particularly important. Using them will allows us to develop a systematic approach for constructing any combinational function.

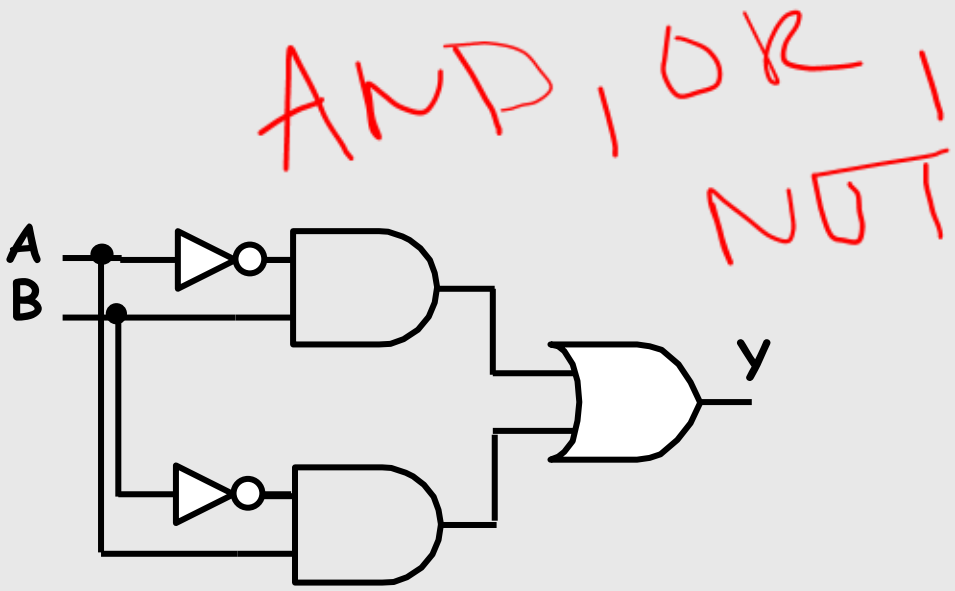
Composing gates

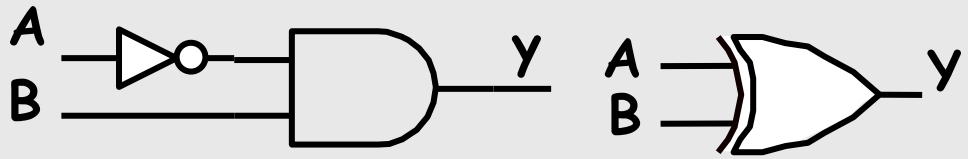
B>A

AB	Y
00	0
01	1
10	0
11	0

XOR

AB	Y
00	0
01	1
10	1
11	0





How many different gates do we really need?

We can always do it with 3 different types of gates (AND, OR, INVERT), and sometimes with 2, but, can we use fewer?

Implementing XOR with AND, OR, NOT!

Truth Table

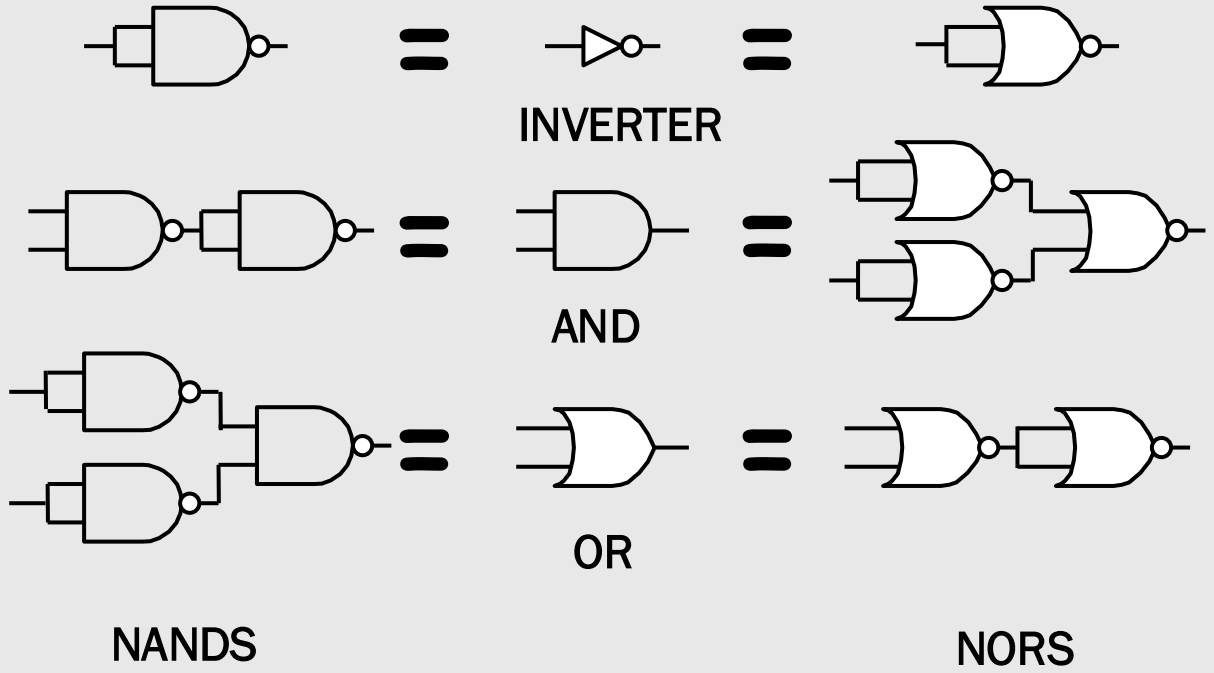
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Challenge question: convert this use ONLY NAND gates. How many NAND gates did you use?

One gate will do!

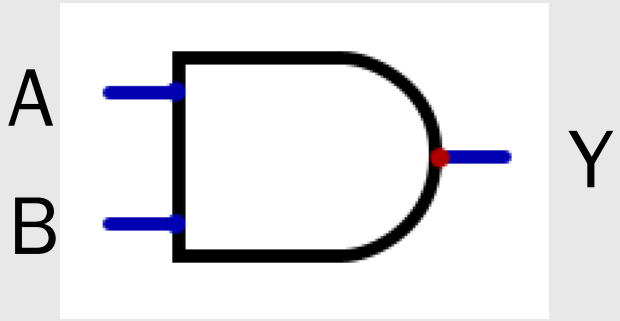
A UNIVERSAL gate is one that can be used to implement *ANY* COMBINATIONAL FUNCTION. There are many UNIVERSAL gates, but not all gates are UNIVERSAL.

NANDs and NORs are *UNIVERSAL*!



AND Gate

Symbol



Truth Table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Equation

$$Y = A \times B$$

or

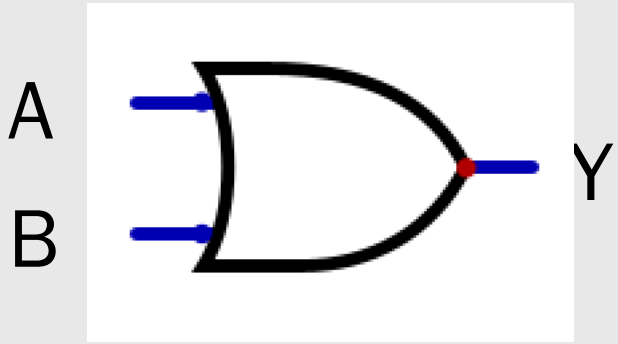
$$Y = AB$$

$Y = A \cdot B$

$A * B$

OR Gate

Symbol



Truth Table

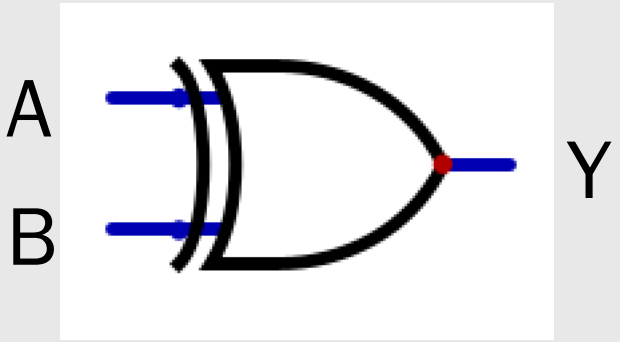
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Equation

$$Y = A + B$$

XOR Gate

Symbol



Truth Table

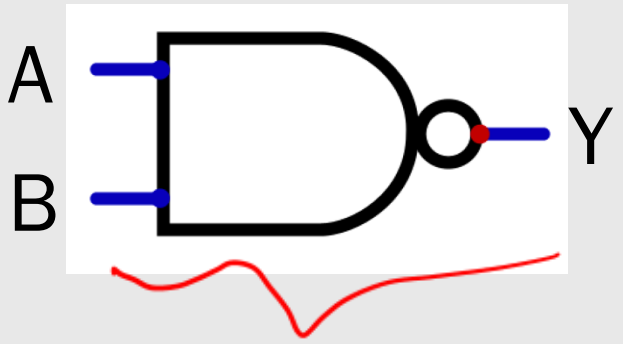
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Equation

$$Y = A \oplus B$$

NAND

Symbol



Truth Table

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Equation

$$Y = \overline{A \; x \; B}$$

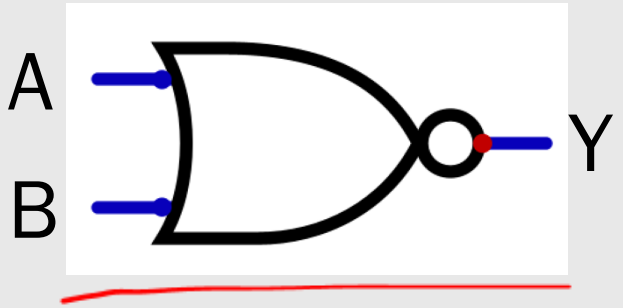
or

$$Y = \overline{AB}$$

$Y = \overline{A \times B}$

NOR

Symbol



Truth Table

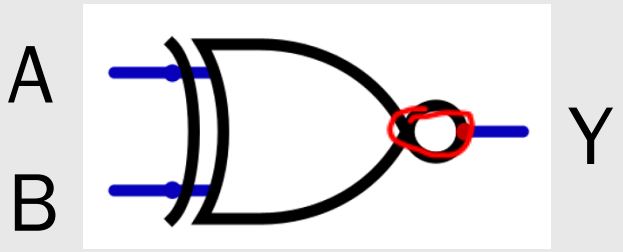
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Equation

$$Y = \overline{A + B}$$

XNOR Gate

Symbol



Truth Table

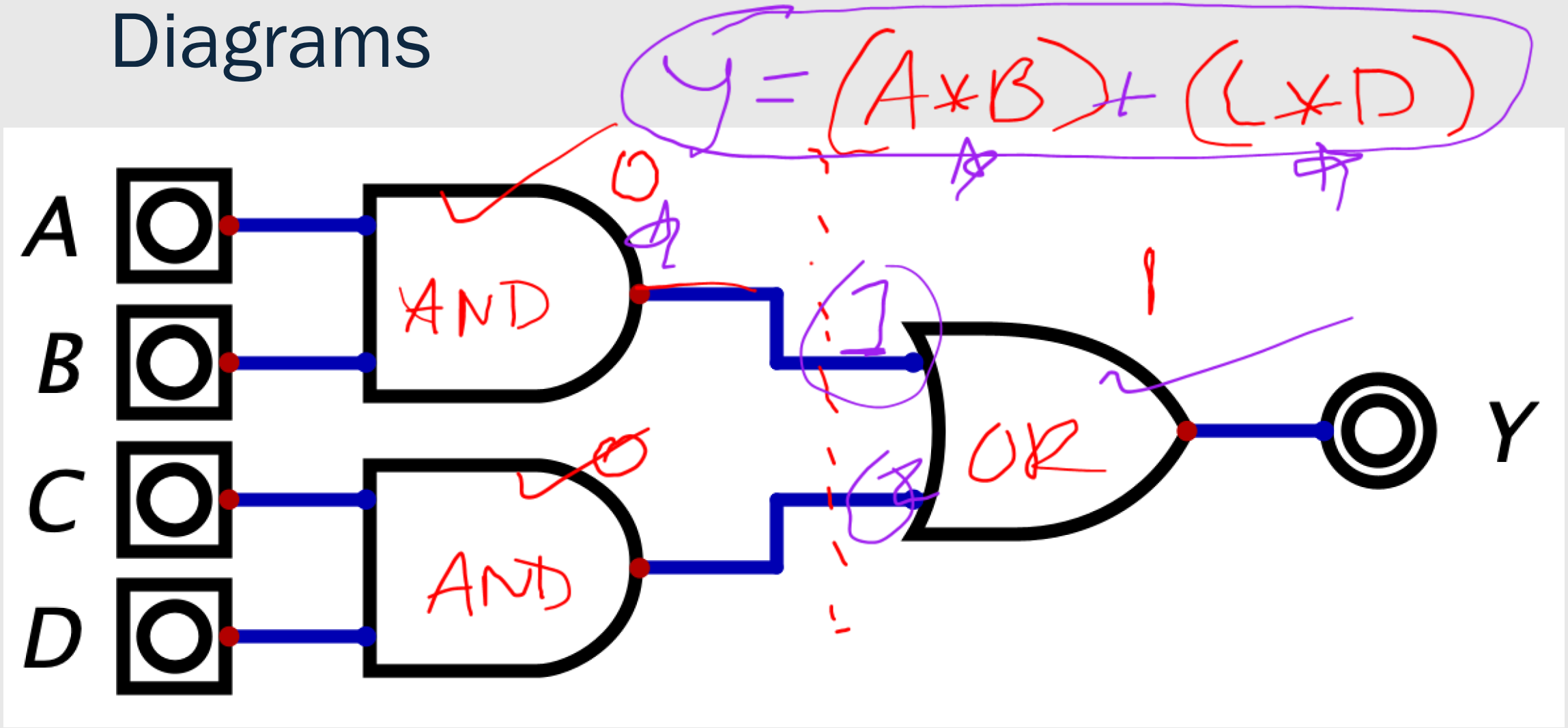
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Equation

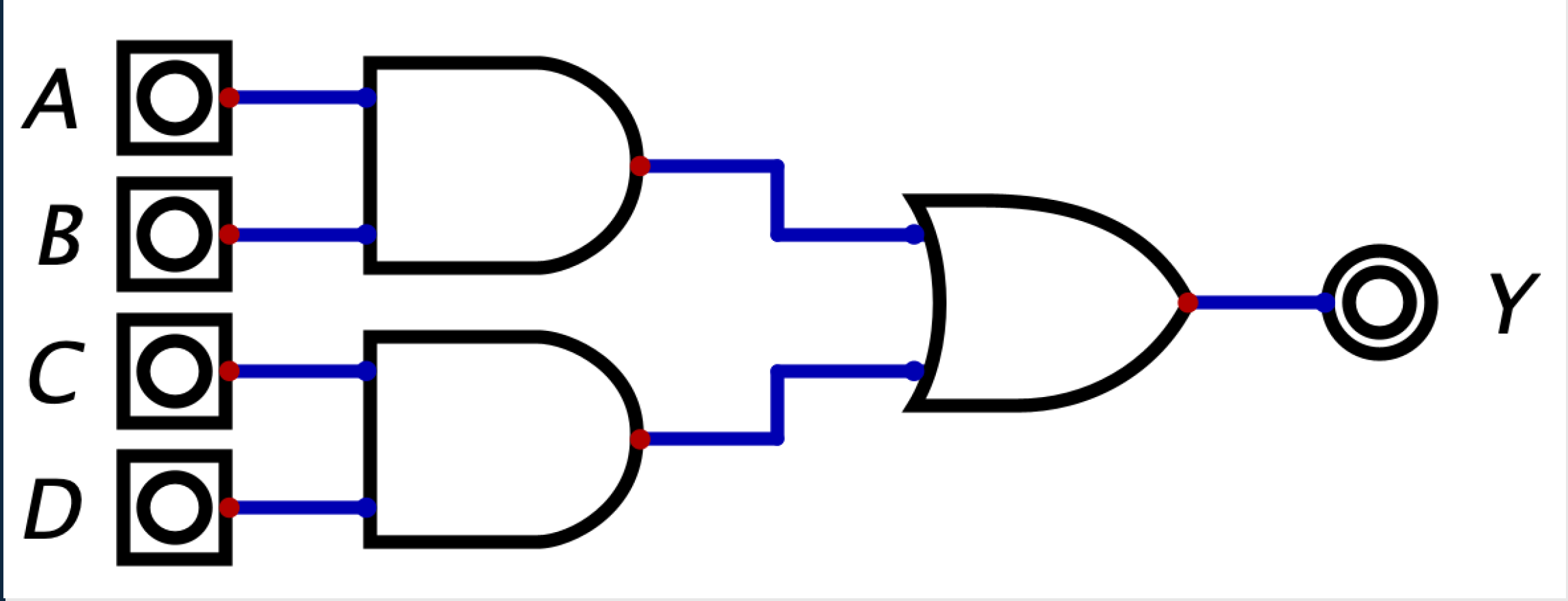
$$Y = \overline{A \oplus B}$$

FORMING EQUATIONS FROM LOGIC DIAGRAMS

Ex1: Forming Equations from Logic Diagrams

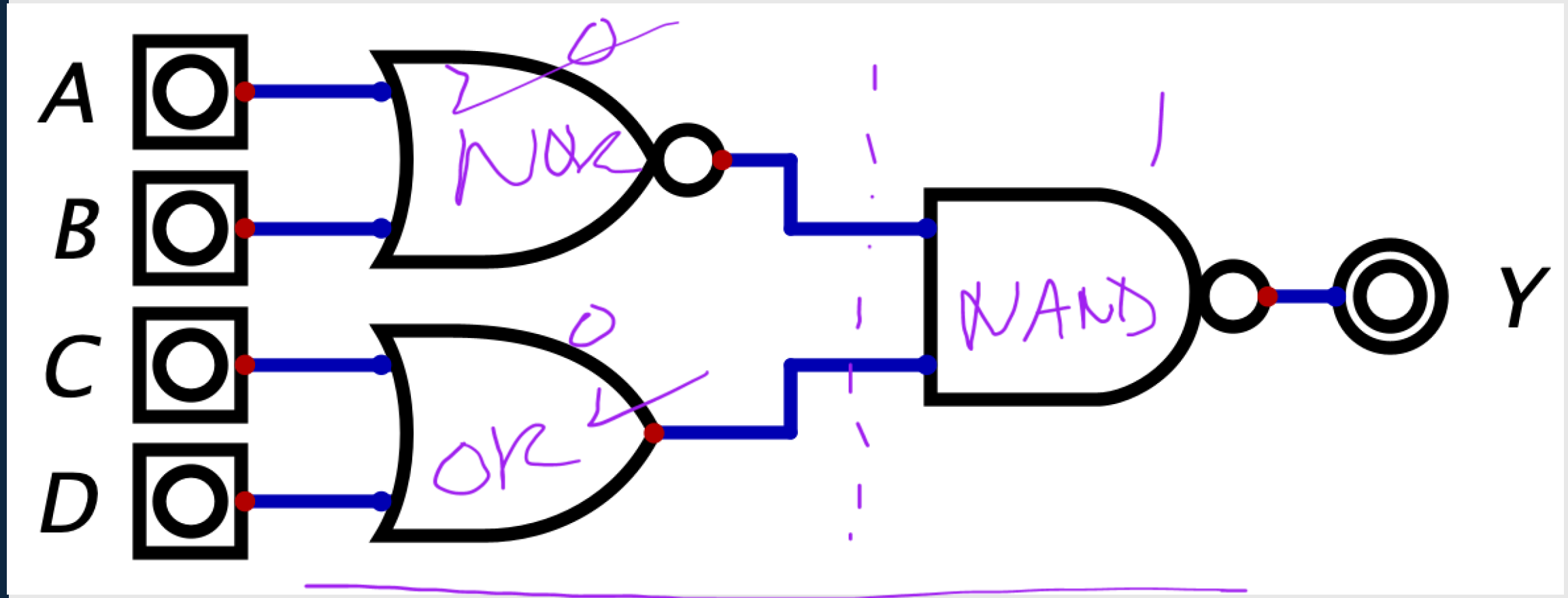


Ex1: Forming Equations from Logic Diagrams



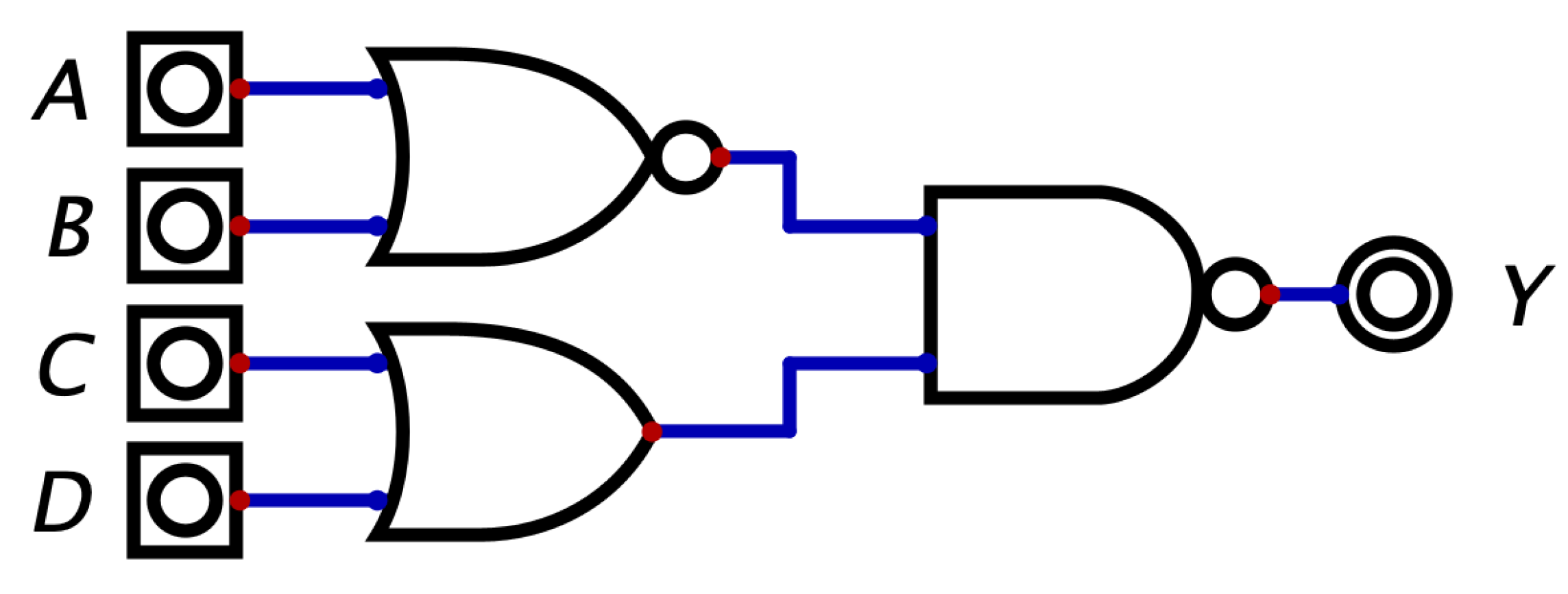
$$Y = AB + CD$$

Ex2: Forming Equations from Logic Diagrams



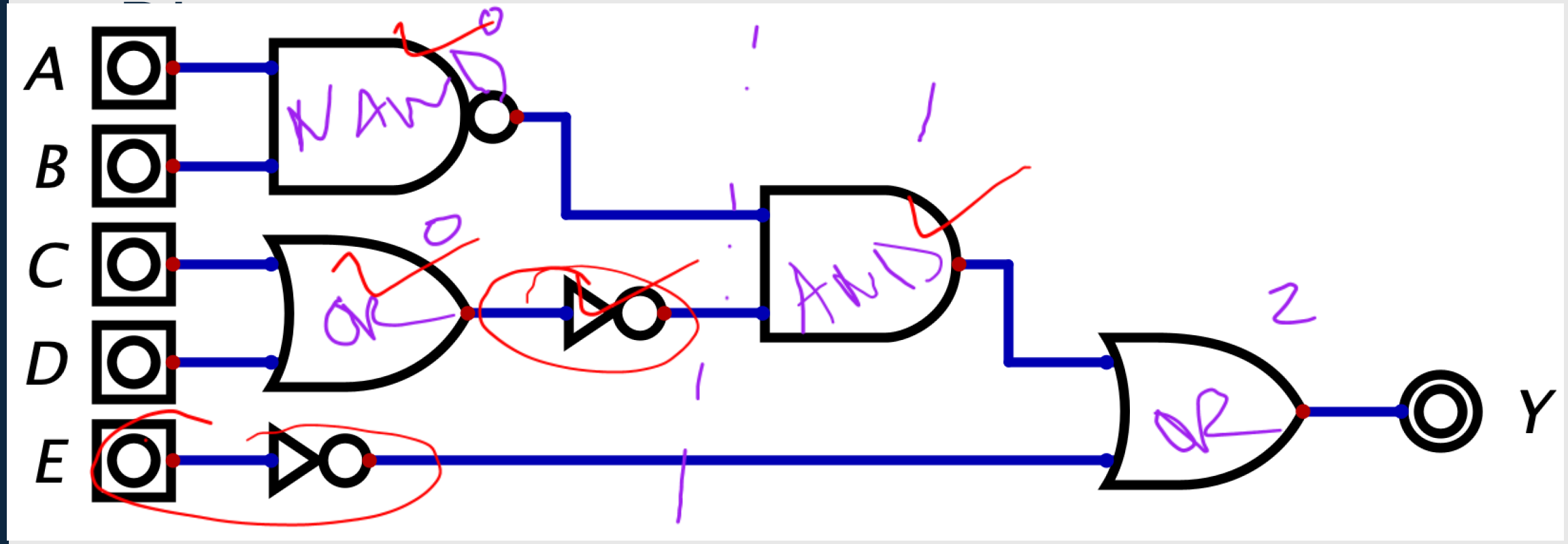
$$Y = \overline{(A+B)} * (C+D)$$

Ex2: Forming Equations from Logic Diagrams



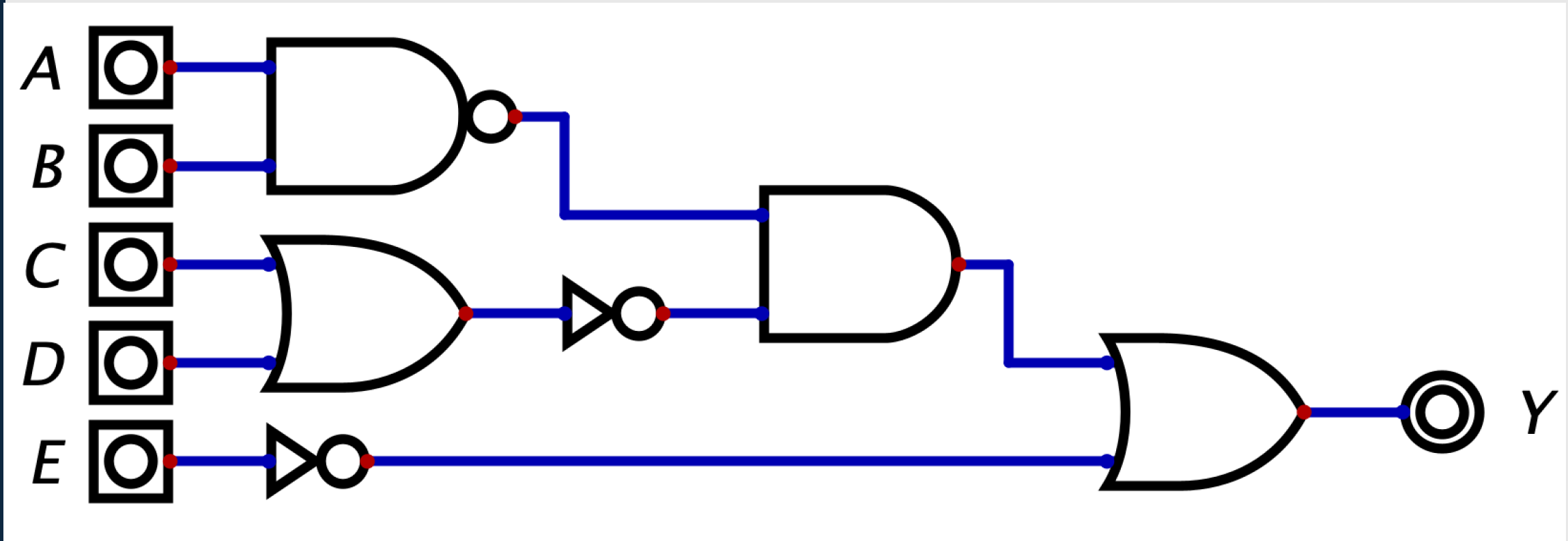
$$Y = \overline{(\overline{A + B})(C + D)}$$

Ex3: Forming Equations from Logic



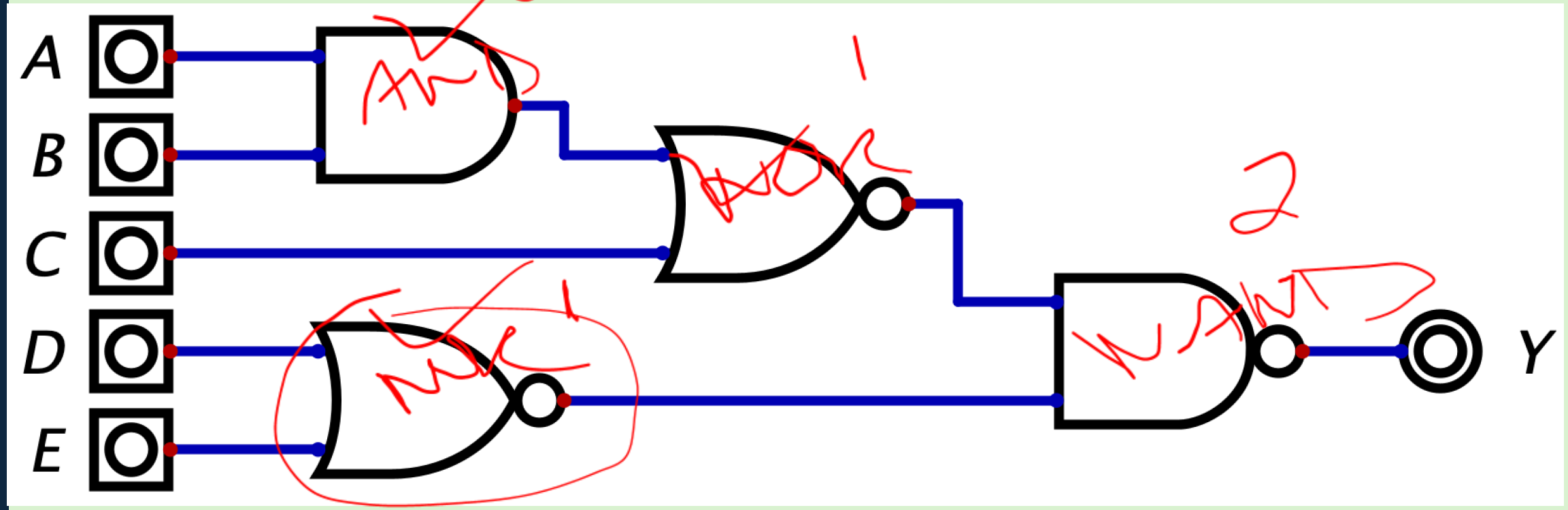
$$Y = \overline{\left[\overline{(A * B)} * \overline{(C + D)}\right]} + \overline{E}$$

Ex3: Forming Equations from Logic Diagrams



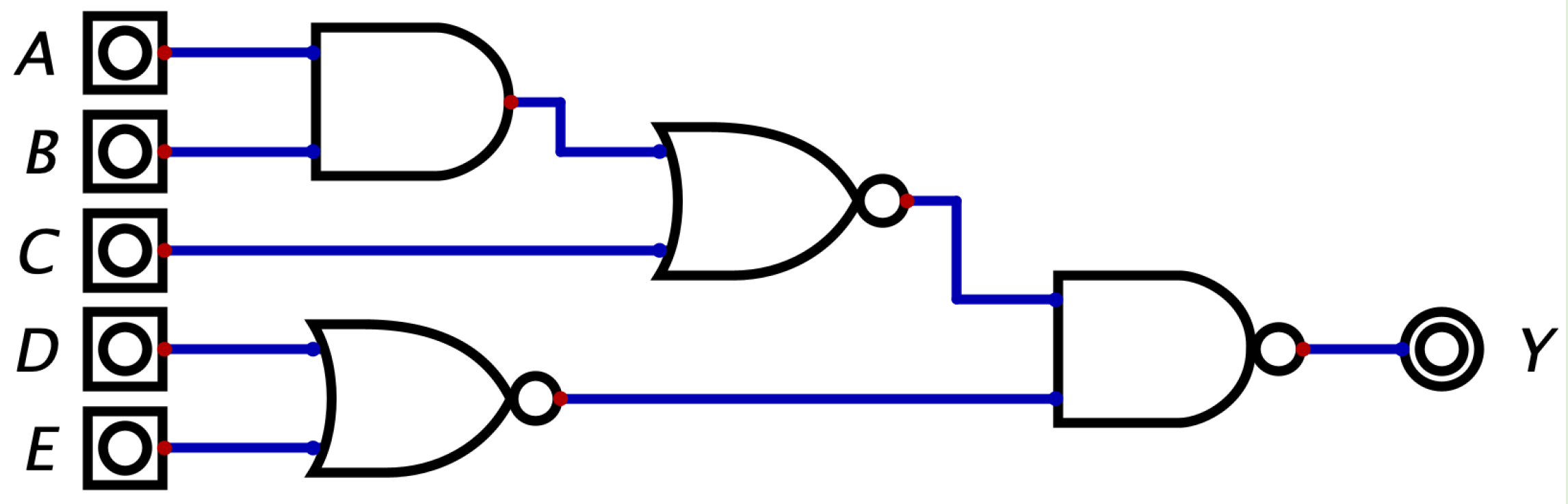
$$Y = (\overline{AB})\overline{(C + D)} + \bar{E}$$

EX4: Forming Equations from Logic Diagrams



$$(\overline{AB+C})\cdot\overline{(D+E)} = Y$$

Forming Equations from Logic Diagrams

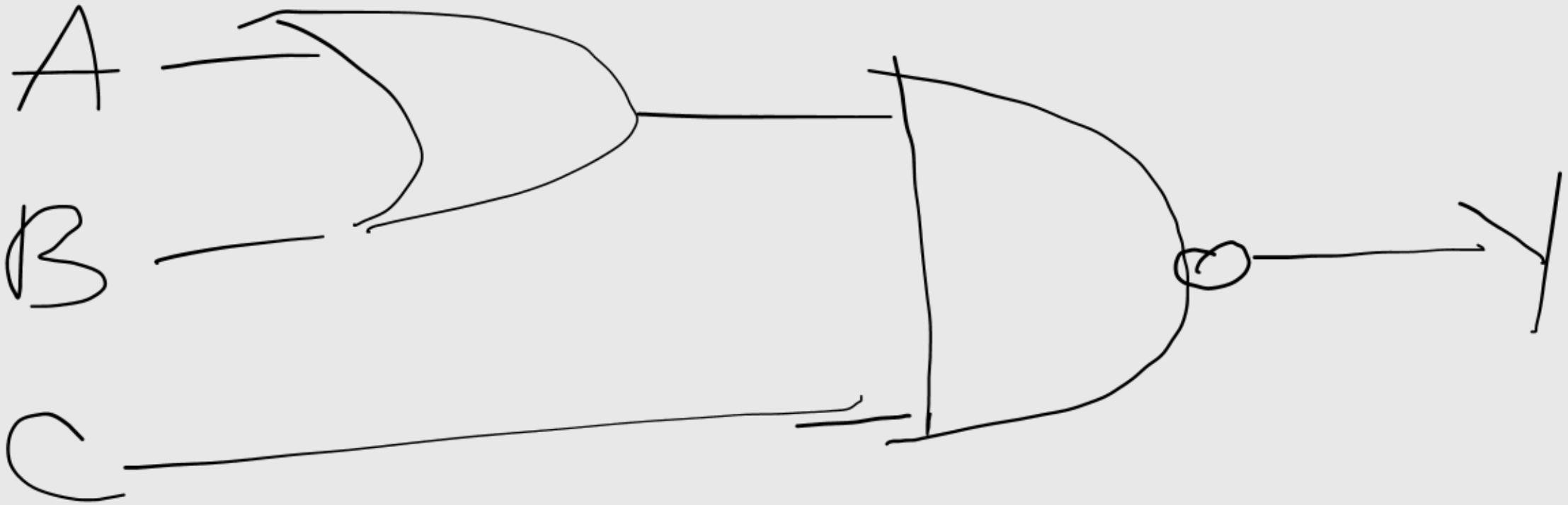


$$Y = \overline{\overline{(AB + C)}\ \overline{(D + E)}}$$

FORMING LOGIC DIAGRAMS FROM EQUATIONS

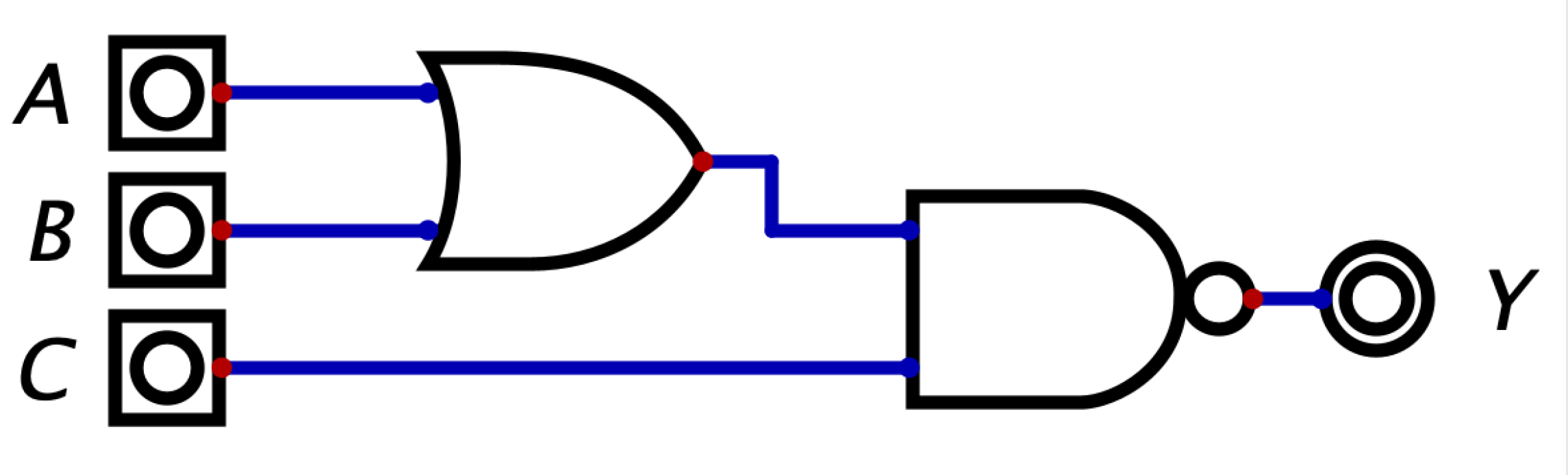
Ex1: Forming Logic Diagrams from Equations

$$Y = \overline{(A + B)C}$$



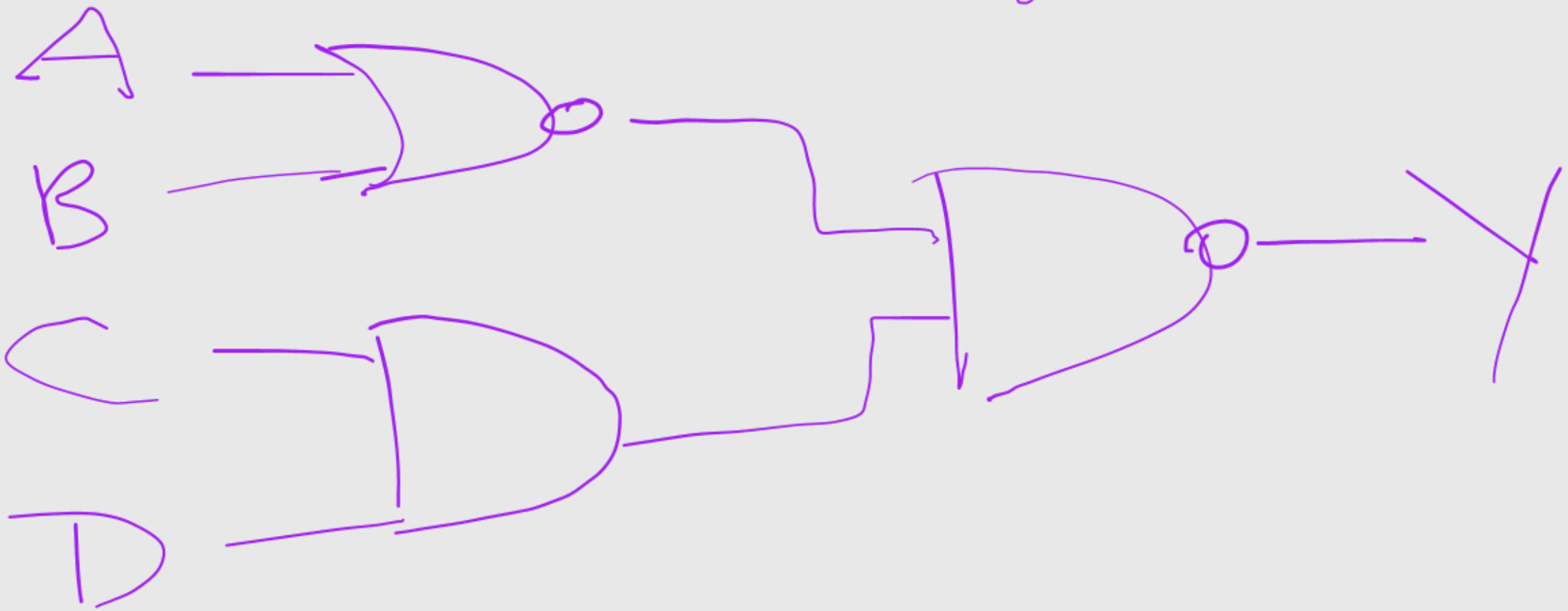
Ex1: Forming Logic Diagrams from Equations

$$Y = \overline{(A + B)C}$$

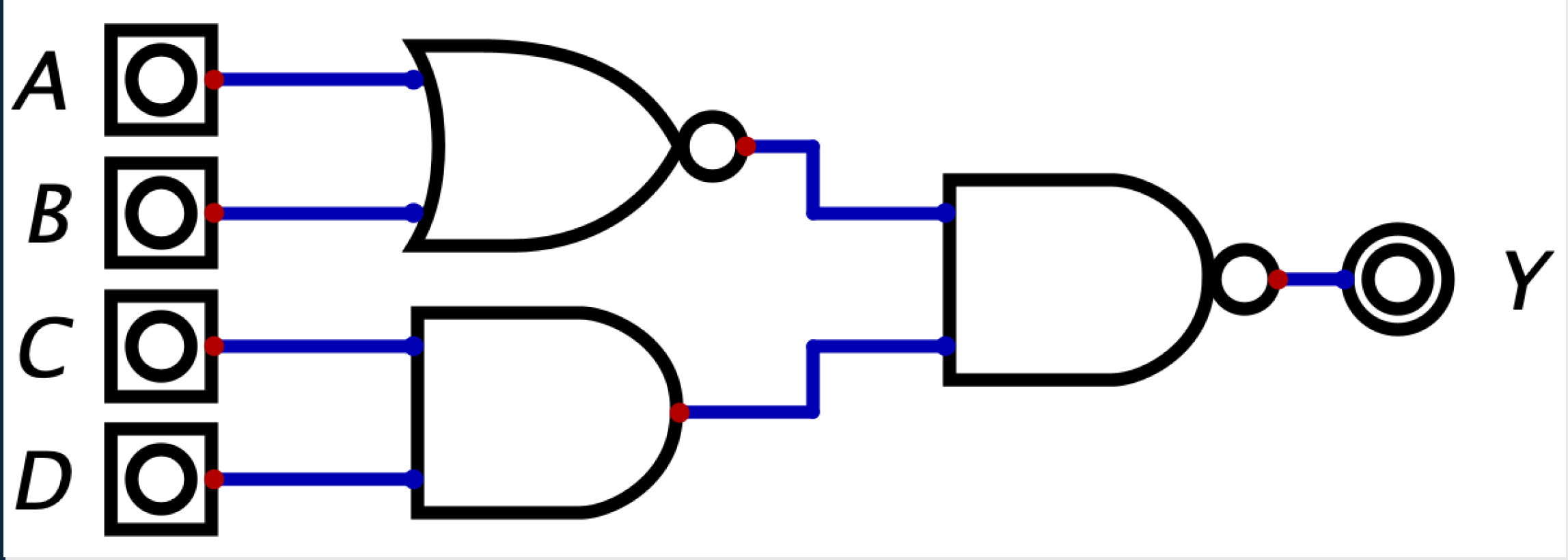


Ex2: Forming Logic Diagrams from Equations

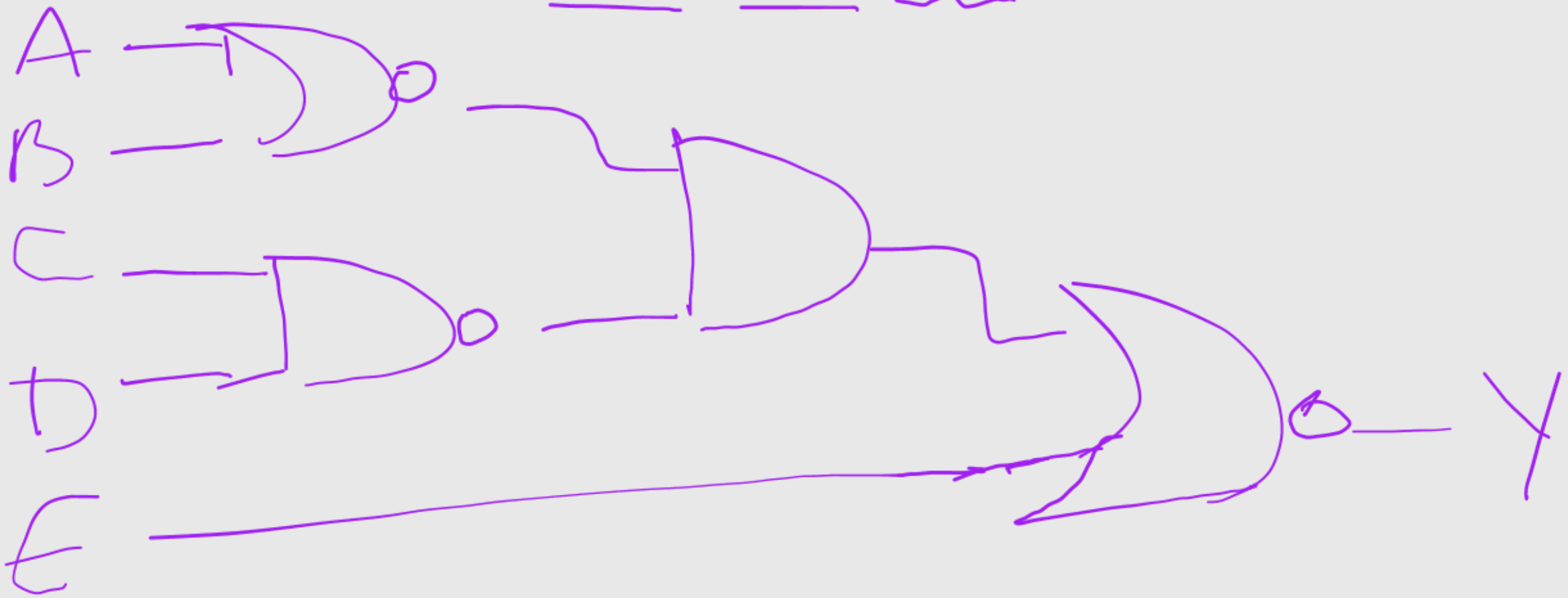
$$Y = \overline{\overline{(A+B)}CD}$$



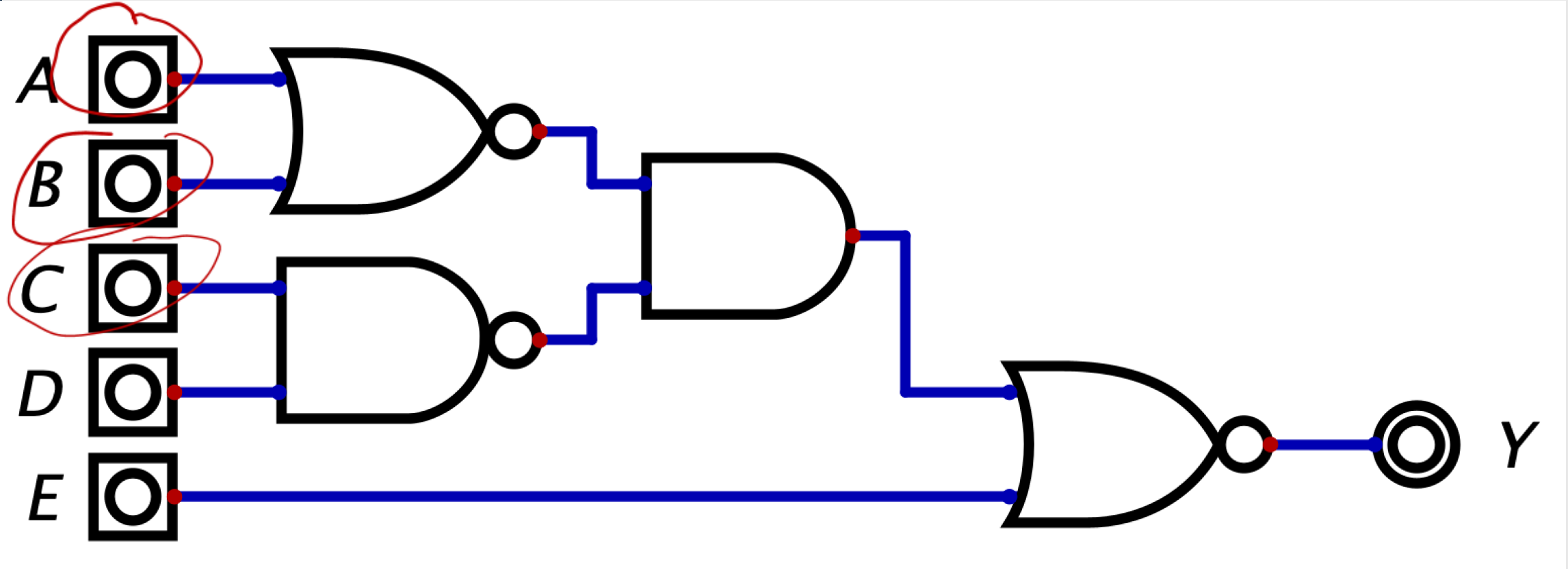
Ex2: Forming Logic Diagrams from Equations $Y = \overline{\overline{(A + B)}CD}$



Ex3: Forming Logic Diagrams from Equations $Y = \overline{(\overline{A + B})(\overline{CD}) + E}$

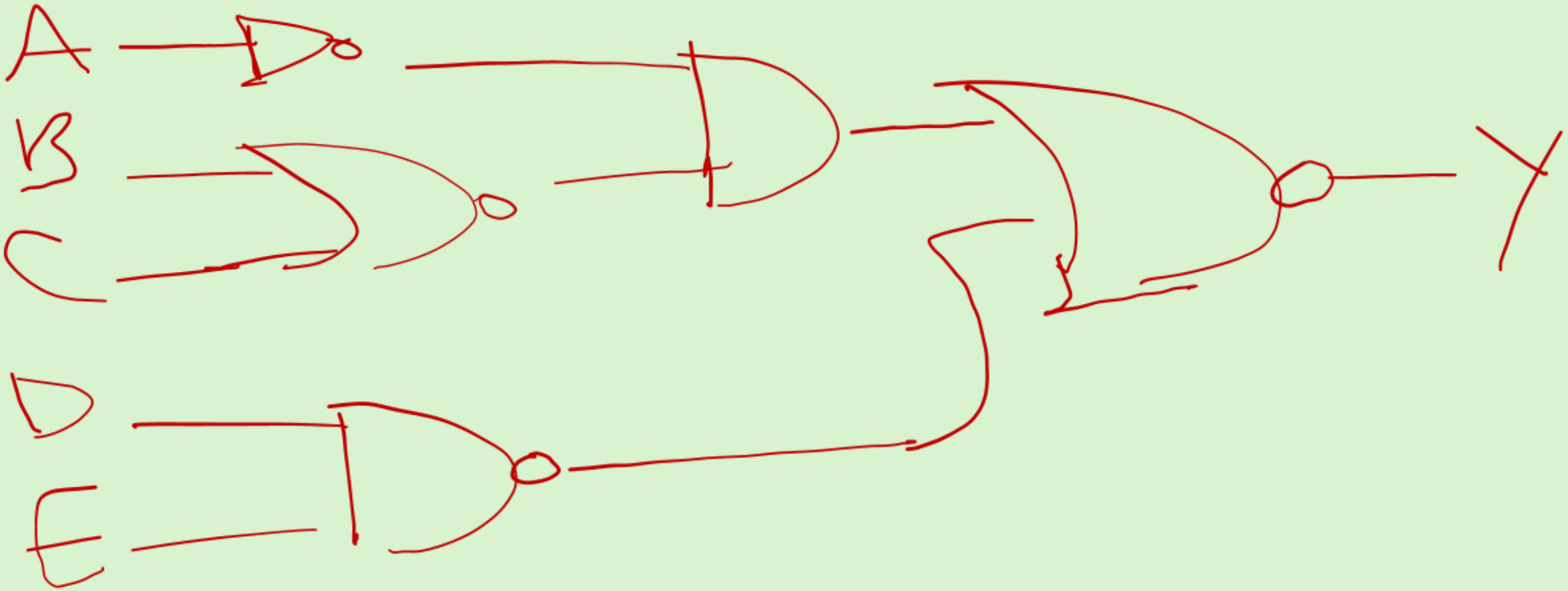


Ex3: Forming Logic Diagrams from Equations $Y = \overline{(\overline{A + B})(\overline{CD}) + E}$



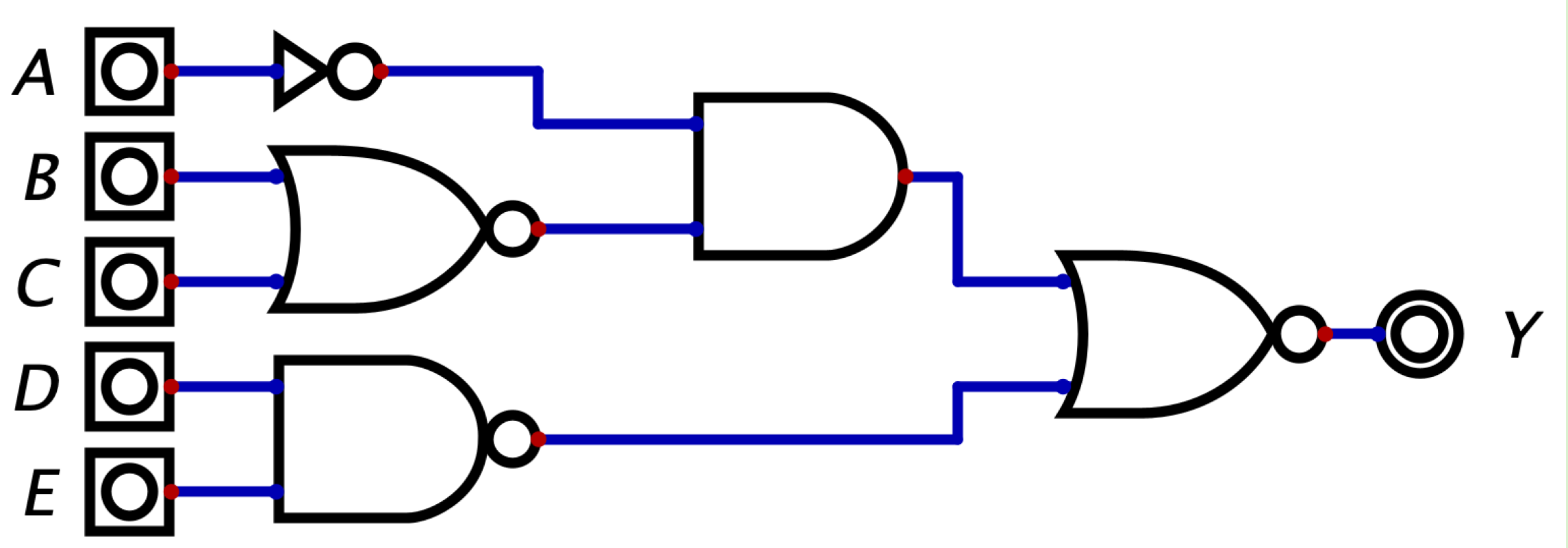
Forming Logic Diagrams from Equations

$$Y = \overline{(\bar{A})(\overline{B + C}) + \overline{DE}}$$



Forming Logic Diagrams from Equations

$$Y = (\bar{A})(\overline{\overline{B + C}}) + \overline{\overline{DE}}$$



FORMING TRUTH TABLES FROM EQUATIONS

Ex1: Forming Truth Tables from Equations

$$Y = A + \bar{B}$$

A	B	Y
0	0	1
0	1	0
1	0	1
1	1	1

Ex1: Forming Truth Tables from Equations

$$Y = A + \bar{B}$$

A	B	Y
0	0	1
0	1	0
1	0	1
1	1	1

Ex2: Forming Truth Tables from Equations

$$Y = AB + \bar{C}$$

A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Ex2: Forming Truth Tables from Equations

$$Y = AB + \bar{C}$$

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Forming Truth Tables from Equations

$$Y = \bar{A}\bar{B} + \overline{AB}$$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

FORMING EQUATIONS FROM TRUTH TABLES

Sum of Products Form

When two or more products (AND) are summed (OR) together

Sum of Products

$Y = AB + AC$

Not Sum of Products

$Y = (A + B)(C + A)$

Approach: write down a boolean expression for every ‘1’ in the output

Ex: Forming Equations from Truth Tables

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Ex: Forming Equations from Truth Tables

A	B	C	Y	
0	0	0	1	$\bar{A}\bar{B}\bar{C}$
0	0	1	0	
0	1	0	1	$\bar{A}B\bar{C}$
0	1	1	0	
1	0	0	0	
1	0	1	1	$A\bar{B}C$
1	1	0	0	
1	1	1	0	

minterms

$$Y = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}C$$

Forming Equations from Truth Tables

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

A Systematic Design Approach

Truth Table

C	B	A	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

-it's systematic!
-it works!
-it's easy!
-we are done! (?)

1) Write the functional spec as a truth table
2) Write down a Boolean expression for every '1' in the output

```
Y = (!C && !B && A) || (!C && B && A)
  || (C && B && !A) || (C && B && A)
```

3) Wire up the ideal gates, replace them with equivalent realizable gates, call it a day, and go home!

This approach will always give us logic expressions in a particular form:

SUM-OF-PRODUCTS

Straightforward Synthesis

We can implement sum-of-products with just 3 levels of logic, using…

INVERTERS/AND/OR!